1. **Test environment**

|  |  |
| --- | --- |
| Hardware Information | SG2042 |
| Architecture | RISC-V64 |
| operating system | openEuler 24.03 (LTS) |
| Python Version | 3.12.2 |
| GCC Version | 12.3.1 |
| G++ Version | 12.3.1 |
| OpenCV version | 4.10.0 |
| Kernel version | 6.6.0 |
| Perf Version | 6.6.0 |

1. **Test Preparation**

Sample address 1: <https://github.com/opencv/opencv/tree/4.x/samples/cpp>

Branch: 4.x

Sample address 2: <https://github.com/opencv/opencv_zoo>

Branch: main

The operation is as follows (build Opencv and install opencv-python according to the build document. For details, see the build document. If you have built OpenCV according to the build document, you can skip it and just set the following opencv\_zoo):

1. OpenCV preparation:
2. Download and build OpenCV and Opencv-python:

|  |
| --- |
| #Install Opencv-python  Pip install numpy  #Download  git clone -b 4.10.0 https://github.com/opencv/opencv.git  cd opencv  #Install additional data  cd sample/data  git clone <https://github.com/opencv/opencv_extra.git>  #Build commands  cmake -D CMAKE\_BUILD\_TYPE=Release \  -D OPENCV\_GENERATE\_PKGCONFIG=ON \  -D CMAKE\_INSTALL\_PREFIX=../../opencv \  -D BUILD\_TIFF=OFF \  -D OPENCV\_ENABLE\_NONFREE=ON \  -D BUILD\_EXAMPLES=ON \  -D BUILD\_SHARED\_LIBS=ON \  -D WITH\_FFMPEG=ON \  -D WITH\_OPENCL=OFF \  -D BUILD\_opencv\_apps=ON \  -D BUILD\_TESTS=ON \  -D INSTALL\_BIN\_EXAMPLES=ON \  -D INSTALL\_C\_EXAMPLES=ON \  -D INSTALL\_PYTHON\_EXAMPLES=ON \  -D INSTALL\_TESTS=ON \  -D CV\_DISABLE\_OPTIMIZATION=ON \  -D WITH\_OPENMP=OFF \  -D ENABLE\_NEON=OFF \  -D OPENCV\_EXTRA\_MODULES\_PATH=../../opencv\_contrib/modules \  -D PYTHON3\_INCLUDE\_DIR=$(python3 -c "from distutils.sysconfig import get\_python\_inc; print(get\_python\_inc())") \  -D PYTHON3\_LIBRARY=$(python3 -c "import distutils.sysconfig as sysconfig; import os; print(os.path.join(sysconfig.get\_config\_var('LIBDIR'), 'libpython' + sysconfig.get\_config\_var('VERSION') + '.so'))") \  -D ENABLE\_OMIT\_FRAME\_POINTER=OFF \  -D CMAKE\_C\_FLAGS="-O0 -g -fPIC -fno-omit-frame-pointer -fno-optimize-sibling-calls -Wall" \  -D CMAKE\_CXX\_FLAGS="-O0 -g -fPIC -fno-omit-frame-pointer -fno-optimize-sibling-calls -Wall" \  -D CMAKE\_INCLUDE\_PATH=/opt/glibc-2.38/include \  -D CMAKE\_EXE\_LINKER\_FLAGS="-L/opt/glibc-2.38/lib" \  -D CMAKE\_BUILD\_RPATH="/opt/glibc-2.38/lib" \  -D CMAKE\_INSTALL\_RPATH="/opt/glibc-2.38/lib" \  -DCMAKE\_EXE\_LINKER\_FLAGS="-Wl,--dynamic-linker=/opt/glibc-2.38/lib/ld-linux- riscv 64.so.1" \  ../ |

1. Compile the test suite:

|  |
| --- |
| #Remember to apply the patch file before compiling and installing  git apply 20240820 .patch  #Compile and install Opencv  make -j$(nproc)  sudo make install |

(3) Run the sample: Enter the /opencv/bin/cpp directory and you will see the successfully compiled sample. For specific commands, see the sample test below.

1. OpenCV \_zoo preparation:
2. Clone the OpenCV\_zoo repository:

|  |
| --- |
| git clone https://github.com/opencv/opencv\_zoo && cd opencv\_zoo  git lfs install  git lfs pull  #Activate the virtual environment and use the virtual environment where opencv is installed  source ./opencv/python3.1 2 \_env /bin/activate  cd benchmark  #Download data  python download\_data.py |

**OpenCV main sample script description:**

The cloning\_demo and cloning\_gui samples are not automated. Please run them by yourself. Before running, you need to copy the testdata/cv/cloning in opencv\_extra to the same directory as the sample. For specific running commands, please refer to the sample instructions

For the train\_HOG example, you need to manually copy the data file to the same directory

|  |
| --- |
| You need to create and copy these images in the same directory as the sample  positive\_images/  Blender\_Suzanne1.jpg  Blender\_Suzanne2.jpg  negative\_images/  basketball1.png  basketball2.png  test\_images/  fruits.jpg  home.jpg  These files are in opencv/samples/data |

Just run the following four scripts in sequence. There is an unknown bug that causes the watershed sample to not be run. The generated logs and csv files should not contain watershed sample data. You need to run and collect data manually.

None of the four scripts require any parameters and can be executed directly.

**1.create\_file.py :**

For the 3calibration test item , a text file input\_data.txt containing the image file path is automatically generated for input data preparation for 3D calibration.   
For the calibration test item, an XML format image list file image\_list.xml is created, which contains the file paths of the left and right groups of images for subsequent image processing and calibration work.

For the digits\_lenet test item, download two files of the Lenet neural network model, namely the .caffemodel and .prototxt files.

For the pca test item , a text file images.txt containing the image file path is generated, which is mainly used for image data input of principal component analysis (PCA) .

For the stereo\_calib test item, an XML file stereo\_calib.xml is created, which contains the file path of the left and right image pairs for binocular stereo vision calibration.

**2. creat\_sample\_config.py** :

sample.conif can generate a script suitable for sexual function analysis. It contains commands for all samples that can be run automatically.

**3. test\_opencv ：**

Performance analysis automation script can achieve fully automated performance acquisition for all but a few samples

The script defines a fixed perf command section and specifies multiple performance events that need to be collected, such as duration\_time, task-clock, cycles, instructions, cache-references, cache-misses, etc. These indicators can fully reflect the performance of command execution, including time consumption, CPU clock cycles, instruction execution status, and cache hit rate.

The script checks the existence of the samples.conf configuration file to ensure that the command list is loaded before execution. If the configuration file does not exist, the script will output an error message and terminate execution to prevent exceptions caused by an empty command list.

In order to organize and store the analysis results, the script will create a results directory in the current directory, and all log files will be saved in this directory. This organization method is conducive to the centralized management of results and subsequent analysis and processing.

The script reads the command pairs (the sample name and the command itself) in the configuration file line by line through a while loop. For each line, comment lines starting with # and blank lines are first skipped to ensure that only valid commands are processed.

For each valid command, the script will construct a complete perf command and append its execution results to the corresponding log file. The log file is named after the sample name and saved in the results directory. Each log file first records the sample name, followed by the perf output results, and finally ends with a separator to distinguish multiple execution processes.

After processing all samples, the script outputs a prompt message to inform the user that all commands have been executed and points out the directory where the result files are located. This design can help users quickly confirm the execution status of all analysis tasks.

**4. genera\_csv.py :**

Performance result parsing and summary script , I wrote a Python script to parse the performance analysis log files generated by the perf tool and summarize the results into a CSV file. The script automatically extracts key performance indicators and calculates IPC (instructions per cycle), thus providing convenient data support for subsequent performance analysis.

**OpenCV\_zoo script description:**

Before running the automated script, copy the mold in the opencv\_zoo root directory to the benchmark directory: cp -r /path/to/opencv\_zoo/models /path/to/opencv\_zoo/benchmark.

And 2.4.data directory under the test data.zip is the opencv\_zoo data file to be placed in the benchmark directory

The script defines a list of 18 deep learning model examples, covering a variety of computer vision tasks such as face detection, object recognition, pose estimation, etc. This diversity enables the script to comprehensively evaluate the performance of different types of models.

In terms of performance indicators, the script specifies 13 key indicators, including execution time, CPU clock cycles, number of instructions, cache access, etc. These indicators are collected through the Linux perf tool and can fully reflect the operating efficiency and resource utilization of the model.

The script uses the subprocess module to execute the perf command and uses an error handling mechanism to ensure that the failure of a single sample does not affect the overall test. This design enhances the robustness and reliability of the script.

For each sample, the script parses the perf output, extracts key performance data, and calculates important derivative metrics such as IPC (instructions per cycle). This automated data processing greatly improves analysis efficiency.

The script uses the tqdm library to display the test progress, allowing users to understand the test status in real time, improving the user experience.

Finally, all test results are integrated into a CSV file for subsequent data analysis and visualization. This output format is both intuitive and easy to further process.

1. **Sample test**

The first seventeen test cases: face\_detection\_yunet, face\_recognition\_sface, facial\_expression\_recognition, handpose\_estimation\_mediapipe, human\_segmentation\_pphumanseg, image\_classification\_mobilenet, image\_classification\_ppresnet, license\_plate\_detection\_yunet, objec t\_detection\_nanodet, object\_detection\_yolox, object\_tracking\_vittrack, palm\_detection\_mediapipe, person\_detection\_mediapipe, person\_reid\_youtureid, pose\_estimation\_mediapipe, qrcode\_wechatqrcode, text\_detection\_ppocr, text\_recognition\_crnn

Run the use case:

|  |
| --- |
| #Replace the red part with the name of the sample to be tested  python benchmark.py --cfg ./config/ face\_detection\_yunet .yaml  #Run all examples  python benchmark.py --all |

Twenty sample running results:

|  |
| --- |
| #1, face\_detection\_yunet  **mean median min input size model**  **30.75 32.69 21.75 [160, 120] YuNet with ['face\_detection\_yunet\_2023mar.onnx']**  **30.02 32.23 21.75 [160, 120] YuNet with ['face\_detection\_yunet\_2023mar\_int8.onnx']**  **#2,** face\_recognition\_sface  **132.45 120.56 115.01 [150, 150] SFace with ['face\_recognition\_sface\_2021dec.onnx']**  **192.21 206.37 115.01 [150, 150] SFace with ['face\_recognition\_sface\_2021dec\_int8.onnx']**  **#3、**facial\_expression\_recognition  **54.54 56.27 53.05 [112, 112] FacialExpressionRecog with ['facial\_expression\_recognition\_mobilefacenet\_2022july.onnx']**  **97.16 104.26 53.05 [112, 112] FacialExpressionRecog with ['facial\_expression\_recognition\_mobilefacenet\_2022july\_int8.onnx']**  **#4、**handpose\_estimation\_mediapipe  **63.94 72.07 60.27 [224, 224] MPHandPose with ['handpose\_estimation\_mediapipe\_2023feb.onnx']**  **72.49 67.93 60.27 [224, 224] MPHandPose with ['handpose\_estimation\_mediapipe\_2023feb\_int8.onnx']**  **#5、**human\_segmentation\_pphumanseg  **231.65 237.28 226.70 [192, 192] PPHumanSeg with ['human\_segmentation\_pphumanseg\_2023mar.onnx']**  **164.70 169.98 107.28 [192, 192] PPHumanSeg with ['human\_segmentation\_pphumanseg\_2023mar\_int8.onnx']**  **#6、**image\_classification\_mobilenet  **79.97 80.94 77.13 [224, 224] MobileNet with ['image\_classification\_mobilenetv1\_2022apr.onnx']**  **86.02 87.63 77.13 [224, 224] MobileNet with ['image\_classification\_mobilenetv2\_2022apr.onnx']**  **87.12 93.78 77.13 [224, 224] MobileNet with ['image\_classification\_mobilenetv1\_2022apr\_int8.onnx']**  **88.00 94.27 77.13 [224, 224] MobileNet with ['image\_classification\_mobilenetv2\_2022apr\_int8.onnx']**  **#7、**image\_classification\_ppresnet  **423.84 456.31 400.10 [224, 224] PPResNet with ['image\_classification\_ppresnet50\_2022jan.onnx']**  **420.00 454.39 397.77 [224, 224] PPResNet with ['image\_classification\_ppresnet50\_2022jan\_int8.onnx']**  **#8、**license\_plate\_detection\_yunet  **136.42 137.87 125.28 [320, 240] LPD\_YuNet with ['license\_plate\_detection\_lpd\_yunet\_2023mar.onnx']**  **161.71 178.72 125.28 [320, 240] LPD\_YuNet with ['license\_plate\_detection\_lpd\_yunet\_2023mar\_int8.onnx']**  **#9、**object\_detection\_nanodet  **308.64 327.36 280.45 [416, 416] NanoDet with ['object\_detection\_nanodet\_2022nov.onnx']**  **330.97 338.44 280.45 [416, 416] NanoDet with ['object\_detection\_nanodet\_2022nov\_int8.onnx']**  **#10、**object\_detection\_yolox  **1610.25 1526.83 1509.44 [640, 640] YoloX with ['object\_detection\_yolox\_2022nov.onnx']**  **1238.24 1315.67 904.11 [640, 640] YoloX with ['object\_detection\_yolox\_2022nov\_int8.onnx']**  **#11、**object\_tracking\_vittrack  **146.08 160.28 139.69 [1280, 720] VitTrack with ['object\_tracking\_vittrack\_2023sep.onnx']**  **#12、**palm\_detection\_mediapipe  **112.31 113.46 110.20 [192, 192] MPPalmDet with ['palm\_detection\_mediapipe\_2023feb.onnx']**  **149.68 153.32 110.20 [192, 192] MPPalmDet with ['palm\_detection\_mediapipe\_2023feb\_int8.onnx']**  **#13、**person\_detection\_mediapipe  **119.02 118.69 116.24 [224, 224] MPPersonDet with ['person\_detection\_mediapipe\_2023mar.onnx']**  **#14、**person\_reid\_youtureid  **358.89 359.34 351.23 [128, 256] YoutuReID with ['person\_reid\_youtu\_2021nov.onnx']**  **361.65 367.65 351.23 [128, 256] YoutuReID with ['person\_reid\_youtu\_2021nov\_int8.onnx']**  **#15、**pose\_estimation\_mediapipe  **135.79 151.86 129.66 [256, 256] MPPose with ['pose\_estimation\_mediapipe\_2023mar.onnx']**  **#16、**qrcode\_wechatqrcode  **41.73 42.20 39.23 [100, 100] WeChatQRCode with ['detect\_2021nov.prototxt', 'detect\_2021nov.caffemodel', 'sr\_2021nov.prototxt', 'sr\_2021nov.caffemodel']**  **#17、**text\_detection\_ppocr  **389.48 383.96 380.30 [640, 480] PPOCRDet with ['text\_detection\_cn\_ppocrv3\_2023may.onnx']**  **388.01 386.43 376.12 [640, 480] PPOCRDet with ['text\_detection\_en\_ppocrv3\_2023may.onnx']**  **453.53 402.94 376.12 [640, 480] PPOCRDet with ['text\_detection\_cn\_ppocrv3\_2023may\_int8.onnx']**  **500.39 519.53 376.12 [640, 480] PPOCRDet with ['text\_detection\_en\_ppocrv3\_2023may\_int8.onnx']**  **#18、**text\_recognition\_crnn  **279.44 255.50 243.43 [1280, 720] CRNN with ['text\_recognition\_CRNN\_CH\_2021sep.onnx']**  **267.90 355.81 243.43 [1280, 720] CRNN with ['text\_recognition\_CRNN\_CN\_2021nov.onnx']**  **239.98 267.00 190.45 [1280, 720] CRNN with ['text\_recognition\_CRNN\_EN\_2021sep.onnx']**  **263.17 229.93 190.45 [1280, 720] CRNN with ['text\_recognition\_CRNN\_CH\_2023feb\_fp16.onnx']**  **255.29 192.79 167.02 [1280, 720] CRNN with ['text\_recognition\_CRNN\_EN\_2023feb\_fp16.onnx']**  **264.04 248.08 167.02 [1280, 720] CRNN with ['text\_recognition\_CRNN\_CH\_2022oct\_int8.onnx']**  **269.67 276.74 167.02 [1280, 720] CRNN with ['text\_recognition\_CRNN\_CN\_2021nov\_int8.onnx']**  **265.75 366.29 167.02 [1280, 720] CRNN with ['text\_recognition\_CRNN\_EN\_2022oct\_int8.onnx']** |

Problem encountered:

|  |
| --- |
| Problems that may occur when running the qrcode\_wechatqrcode sample  Problem log: " Traceback (most recent call last):  File "/home/chenweijia/opencv\_zoo/benchmark/benchmark.py", line 13, in <module>  assert opencv\_python\_version(cv.\_\_version\_\_) >= opencv\_python\_version("4.10.0"), \  ^^^^^^^^^^^^^^  AttributeError: module 'cv2' has no attribute '\_\_version\_\_' "  This error indicates that the \_\_version\_\_ attribute is missing from the cv2 module. Normally, the cv2 module should contain a \_\_version\_\_ attribute that returns the version number of OpenCV.  Possible causes:  Incomplete or corrupted OpenCV installation: causing the cv2 module to be missing some critical attributes.  Incorrect OpenCV packages: You may have opencv-python-headless installed, but not the full opencv-python or opencv-contrib-python packages. |

Problem encountered:

|  |
| --- |
| #Uninstall existing OpenCV installation  pip uninstall opencv-python opencv-python-headless opencv-contrib-python  # Clear cache (optional) and reinstall OpenCV  pip cache purge  pip install opencv-contrib-python |

## Test case 18 : 3calibration

This code implements a program for joint calibration of three cameras. It calibrates three horizontally arranged cameras by detecting checkerboard corners, and finally generates the camera's intrinsic and extrinsic parameters. When running, the program first reads the image list provided by the user and detects the checkerboard corners in each image. Then, each camera is calibrated separately, and the stereo camera is calibrated on this basis to calculate the rotation matrix and translation vector between the cameras. After the calibration is completed, the program generates a remapping table for image correction and saves the results to an output file. The program can also display the corrected image to facilitate the user to check the calibration effect.

You need to create the input\_data.txt file in the sample directory with the following content:

|  |
| --- |
| #The number of images should be a multiple of 3, otherwise an error will be reported. There is no left10 because the image cannot be used  ../../opencv/samples/data/left01.jpg  ../../opencv/samples/data/left02.jpg  ../../opencv/samples/data/left03.jpg  ../../opencv/samples/data/left04.jpg  ../../opencv/samples/data/left05.jpg  ../../opencv/samples/data/left06.jpg  ../../opencv/samples/data/left07.jpg  ../../opencv/samples/data/left08.jpg  ../../opencv/samples/data/left09.jpg  ../../opencv/samples/data/left11.jpg  ../../opencv/samples/data/left12.jpg  ../../opencv/samples/data/left13.jpg |

Run the use case:

|  |
| --- |
| ./example\_cpp\_3calibration -w=9 -h=6 -s=1 -o=camera\_params.yml input\_data.txt |

Some results of running:

|  |
| --- |
| Running calibration ...  Camera 1 calibration reprojection error = 0.0391389  Camera 2 calibration reprojection error = 0.0380287  Camera 3 calibration reprojection error = 0.0363169  Pair (1,2) calibration reprojection error = 0.344629  Pair (1,3) calibration reprojection error = 0.324685  Disparity ratio = 0.728765 |

Write the camera parameters into camera\_params.yml

## Test case 19: application\_trace

This code demonstrates how to use OpenCV's application tracking function to process video frames. When the program runs, it first parses the command line parameters to get the video file path to process and the number of frames to process , but I modified the code to use the specified video directly . Then, it tries to open the video file and get basic information about the video, such as width, height, and total number of frames.

After entering the processing loop, the program reads the video frame by frame and converts each frame into a grayscale image, and then applies the Canny edge detection algorithm for processing. Each major operation step (such as frame reading and image processing) is marked using OpenCV's tracing macros (CV\_TRACE\_FUNCTION and CV\_TRACE\_REGION), allowing the performance of each function and code area to be tracked when debugging or profiling.

After processing, the program exits. Throughout the process, if the user specifies the number of frames, the program will process the specified number of frames; otherwise, it will process the entire video until the end. Finally, the program will not display the processed image, nor will it save any results. It is only used to demonstrate how to combine OpenCV's tracking tools for performance analysis.

Run the use case:

|  |
| --- |
| ./example\_cpp\_application\_trace ../../samples/data/vtest.avi |

Operation results:

|  |
| --- |
| Video ../../samples/data/vtest.avi: width=768, height=576, nframes=795  Start processing...  Press ESC key to terminate |

## Test case 20: aruco\_dict\_utils

This code implements a tool for generating and evaluating custom ArUco marker dictionaries and calculating their Hamming distances. When run, the program first parses the command line arguments to determine whether to use a predefined dictionary, a custom dictionary file, or generate a new custom dictionary. Depending on the user's choice, the program will load or generate the appropriate dictionary. If the output file path and dictionary arguments are specified, the program will generate a new dictionary and save it to the output file.

Then, the program calculates the minimum Hamming distance of the tokens in the dictionary according to the user's options. This distance represents the minimum difference between any two tokens in the dictionary. The user can choose whether to consider the flipping (symmetry) effect of the tokens. If symmetry is considered, the program will calculate the distance between the flipped tokens.

At the end of the program, the calculated minimum Hamming distance is printed on the console. The whole process helps to generate a dictionary of ArUco markers with good discrimination in order to improve the reliability of marker detection in computer vision applications.

Run the use case:

|  |
| --- |
| #Use the predefined dictionary DICT\_4X4\_50  ./example\_cpp\_aruco\_dict\_utils -d=DICT\_4X4\_50 output\_dict.yml |

Operation results:

|  |
| --- |
| Hamming distance: 4 |

Test case 22: aruco\_dict\_utils

Run the use case:

|  |
| --- |
| #Use the predefined dictionary DICT\_4X4\_50  ./example\_cpp\_aruco\_dict\_utils -d=DICT\_4X4\_50 output\_dict.yml |

Operation results:

|  |
| --- |
| Hamming distance: 4 |

## Test case 21: asift

This code demonstrates how to use OpenCV for affine feature detection, description, and matching. The program allows the user to select a specific feature detector (such as SIFT, ORB, or BRISK) and matching method (such as Flann or BruteForce) and apply it to two input images. When run, the program first parses the command line arguments to determine the feature detector and matching method to use, and loads the two images to be compared.

Next, the program uses the specified feature detector to extract key points and descriptors from the image and inputs them into the matcher to find the correspondence between the two images. Using the RANSAC method, the program calculates the homography matrix between the matching points and uses the matrix to filter out the inliers (i.e., reliable matching points). The program then visualizes the matching results, including drawing matching points and lines in the image.

Finally, the program saves the visualization result as an image file affine\_find\_obj\_result.jpg and outputs the matching execution time, number of internal points and other related information in the console. The whole process aims to show how to compare the similarity of two images through affine feature matching and apply this technology in practice.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Operation results:

|  |
| --- |
| Hamming distance: 4 |

## Test case 22: audio\_spectrogram (cannot achieve expected function)

This code implements an audio visualization tool that can obtain audio data from a file or microphone and generate the corresponding amplitude graph or spectrum graph. The program supports both static and dynamic drawing modes. Users can specify the input type, drawing type, image type, audio source file and other related parameters through command line parameters.

When running, the program first parses the command line parameters to determine the source of the input audio (file or microphone), the type of drawing (static or dynamic), the type of image (amplitude graph, spectrogram, or a combination of both), and other configuration parameters. For static drawing, the program reads the entire audio file or the audio data recorded by the microphone, generates a complete amplitude graph or spectrogram, and finally saves the result as an image file. If dynamic drawing is selected, the program will continuously update the audio data within the time window, generating and updating the image in real time.

The program implements a variety of audio processing and image drawing functions, including reading audio files or microphone input, short-time Fourier transform (STFT), drawing amplitude graphs and spectrograms, and image splicing and saving. The image files generated in static mode will be saved as separate files according to the user's choice, such as "amplitude\_graph.png" or "spectrogram.png"; in dynamic mode, the image will be continuously updated and finally saved as the corresponding file, such as "dynamic\_amplitude\_graph.png" or "dynamic\_spectrogram.png".

At the end of the program, all generated images will be saved in the specified directory, and users can view the image save path in the console. Through this code, users can intuitively view the amplitude and spectrum information of the audio signal, which is convenient for audio analysis and visualization.

Run the use case:

|  |
| --- |
| ./example\_cpp\_audio\_spectrogram -i=file -d=static -g ampl\_and\_spec -a=Megamind.avi -s=1 -t=Rect -l=256 -o=128 –enableGrid=false -r=400 -c=900 -x=5 -y=5 -z=5 -m=20 -f=5 -u=1 -w=10 |

运行结果：

|  |
| --- |
| CAP\_PROP\_AUDIO\_DATA\_DEPTH: CV\_8U  CAP\_PROP\_AUDIO\_SAMPLES\_PER\_SECOND: 0  CAP\_PROP\_AUDIO\_TOTAL\_CHANNELS: 0  CAP\_PROP\_AUDIO\_TOTAL\_STREAMS: 0  terminate called after throwing an instance of 'cv::Exception'  what(): OpenCV(4.10.0-dev) /home/chenweijia/opencv/modules/core/src/copy.cpp:320: error: (-215:Assertion failed) channels() == CV\_MAT\_CN(dtype) in function 'copyTo'  Aborted (core dumped) |

问题分析：

|  |
| --- |
| 1. Audio stream extraction failed: According to the log provided by OpenCV, the relevant attributes of the audio stream are all zero. This may be because OpenCV cannot correctly parse or process the audio stream, or the audio stream encoding format is not supported.  2. Assertion failure in copyTo function: OpenCV crashes due to mismatch in the number of channels when processing audio data. This may be a subsequent error caused by the failure of audio stream extraction in the previous step. |

## Test case 23: barcode

This code implements an application that uses the OpenCV library to detect and decode barcodes. Users can choose to detect barcodes in real time from a camera or from an image file. The program supports two modes: detection only mode and detection and decoding mode. Through command line parameters, users can specify the input image path, the output result path, whether to detect only barcodes, and whether to use a super resolution model to improve image quality.

When running, if the user specifies an input image path, the program will load the image and perform multiple barcode detection and decoding operations on the image to measure the performance (frame rate). After processing, the program will save the detection results to the specified output file. If the user does not specify an input image path, the program will open the camera and perform real-time barcode detection and decoding.

During the processing of each frame of the image, the program will call the BarcodeDetector class to detect the barcode and decide whether to decode the barcode information based on the mode selected by the user. After the processing is completed, the program will visualize the detected barcode position and decoded information (although the visualization code here is commented out) and print the frame rate information in the console.

At the end of the program, if running in image mode, the resulting image will be saved to the specified file. In camera mode, the user can switch detection modes by pressing the 'd' key, or exit the program by pressing the 'ESC' key. The entire application is designed to show how to use OpenCV's barcode detection function, while supporting super resolution to improve image quality.

Run the use case:

|  |
| --- |
| ./example\_cpp\_barcode --in=../../samples/data/board.jpg --out=../../samples/data/board\_result.jpg --detect=false |

Operation results:

|  |
| --- |
| Image size: [640 x 480]  Mode is <detectAndDecode>  FPS: 17.7663  Saving result: ../../samples/data/board\_result.jpg  Press any key to exit ... |

## Test case 24: bgfg\_segm

This code implements an example program for background segmentation using OpenCV. The user can choose to take input from a camera or a video file through command line parameters, and select different background segmentation algorithms (such as KNN or MOG2). The program processes the video frame, generates a foreground mask through the selected background subtraction algorithm, and saves the processed frame, foreground image, and background image to disk.

When run, the program first opens the camera or video file based on the input source specified by the user, and then creates a background subtraction model. For each frame, the program resizes it and generates a foreground mask using the background subtraction model. The foreground mask can be smoothed based on the user's choice. The processed foreground and background images are saved to the specified directory along with the original frames.

The program supports dynamic adjustment of model update and foreground mask smoothing switches (in the commented-out interactive code). Finally, the program saves all processing results as image files and outputs the save path in the console.

运行用例：

|  |
| --- |
| ./example\_cpp\_bgfg\_segm --fn=../../samples/data/vtest.avi --method=mog2 |

运行结果：

|  |
| --- |
| Saved frame to bgfg\_segm/frame\_0.png  Saved foreground to bgfg\_segm/foreground\_0.png  Saved background to bgfg\_segm/background\_0.png  Saved frame to bgfg\_segm/frame\_1.png  Saved foreground to bgfg\_segm/foreground\_1.png  Saved background to bgfg\_segm/background\_1.png  Saved frame to bgfg\_segm/frame\_2.png  Saved foreground to bgfg\_segm/foreground\_2.png  Saved background to bgfg\_segm/background\_2.png  Saved frame to bgfg\_segm/frame\_3.png  Saved foreground to bgfg\_segm/foreground\_3.png  Saved background to bgfg\_segm/background\_3.png  Saved frame to bgfg\_segm/frame\_4.png  Saved foreground to bgfg\_segm/foreground\_4.png  Saved background to bgfg\_segm/background\_4.png |

## Test case 25: calibration

This code implements a camera calibration tool, where users can calibrate the camera using different checkerboard or circular grid patterns. The code supports real-time capture of images from the camera or reading images from an image list for processing. Through calibration, the code can calculate the intrinsic and extrinsic parameters of the camera and generate a corrected image.

The program supports specifying various calibration settings through command line parameters, including chessboard size, pattern type, number of frames, delay, output file, etc. During the calibration process, the program detects corners or dots in the input image and calculates their corresponding positions in the real world. Finally, the camera's intrinsic parameter matrix and distortion coefficients are generated through the optimization process and saved to the specified file.

After the calibration is complete, the user can choose to dedistort the image and save the corrected image. The entire process can be automated and output the relevant calibration results and error information.

Run the use case:

|  |
| --- |
| ./example\_cpp\_calibration -w=9 -h=6 -s=0.025 -o=camera.yml -op -oe image\_list.xml |

Operation results:

|  |
| --- |
| Use K3 distortion coefficient? 0 |

## Test case 26: camshiftdemo

This code implements a target tracking application based on the CamShift algorithm. The user can select the object to be tracked by the mouse, and the program will use the color histogram of the selected area as the target model and continue to track the object in subsequent frames. The program supports reading input images from multiple image files and performing target tracking on each image. Specific functions include: Image input: Specify a comma-separated list of image files through command line parameters, and the program will read these images in turn for processing. Target selection: The user can select the target area of interest by clicking and dragging with the mouse. Once the selection is completed, the program will calculate the color histogram of the area and initialize the CamShift algorithm for tracking. Color histogram display: The program can display the color histogram of the selected target area to visualize the color distribution of the target model. CamShift algorithm tracking: The program uses the CamShift algorithm to track the target in each frame, updates the position of the target according to the back projection of the color histogram, and marks the tracked area with an ellipse.

Image saving: The processed image and histogram image will be saved as output\_<index>.png and histogram\_<index>.png respectively. User interaction: The program supports some simple keyboard interaction operations, such as switching the back projection mode, clearing the tracking target, showing or hiding the histogram, pausing and continuing the processing, etc. The program provides detailed help information. Users can run the program through the command line and view the corresponding help prompts. Through these functions, users can easily track the selected target in a static image or image sequence and generate the corresponding result image.

Run the use case:

|  |
| --- |
| ./example\_cpp\_camshiftdemo left01.jpg,left02.jpg,left03.jpg,left04.jpg,left05.jpg |

Operation results:

|  |
| --- |
| Hot keys:  ESC - quit the program  c - stop the tracking  b - switch to/from backprojection view  h - show/hide object histogram  p - pause video  To initialize tracking, select the object with mouse  [ WARN:0@0.012] global samples.cpp:61 findFile cv::samples::findFile('left01.jpg') => '/home/chenweijia/opencv/bin/cpp/../../samples/data/left01.jpg' |

## Test case 27: cloning\_demo

This program is an example for image processing, which demonstrates different cloning and image processing techniques in the OpenCV library. The program first lists six image processing techniques for the user to choose from, and the user enters the corresponding number to select one of the techniques. Based on the user's selection, the program loads the corresponding source image, target image, and mask image, and then performs specific image processing operations such as seamless cloning, color change, lighting change, etc. After the processing is completed, the resulting image will be saved to a file.

Specifically, the program flow is as follows:

1. After the user runs the program, he will first see a list of technical options, including six technologies: "Normal Cloning", "Mixed Cloning", "Monochrome Transfer", "Local Color Change", "Local Illumination Change" and "Texture Flattening".
2. The user enters the number of the corresponding option (1-6), and the program loads the corresponding source image, target image, and mask image based on the selected technology.
3. The program will check whether the loaded images are successful. If any images cannot be loaded, the program will output an error message and exit.
4. Depending on the technique selected by the user, the program will call the corresponding OpenCV function to process the image. For example, if "Normal Cloning" is selected, the program will call the seamlessClone function and save the result as a new image file.
5. After processing is completed, the program will save the resulting image as a specified file (such as cloned\_normal.png, cloned\_mixed.png, etc.) and end the run.

The program does not display the processed image at the end, but only saves the result to disk. Therefore, the user needs to view the saved image file to see the processing results.

the testdata/cv/cloning in opencv\_extra to the same directory as the sample.

Run the use case:

|  |
| --- |
| ./example\_cpp\_cloning\_demo |

Run results ( need to enter the run options manually! ):

|  |
| --- |
| Note: specify OPENCV\_SAMPLES\_DATA\_PATH\_HINT=<opencv\_extra>/testdata/cv  Cloning Module  ---------------  Options:  1) Normal Cloning  2) Mixed Cloning  3) Monochrome Transfer  4) Local Color Change  5) Local Illumination Change  6) Texture Flattening  Press number 1-6 to choose from above techniques: 1 |

## Test case 28: cloning\_gui

This program is an image processing demonstration that shows how to use OpenCV's seamless cloning module. The program allows the user to choose six different image processing techniques, including normal cloning, hybrid cloning, monochrome conversion, color change, lighting change, and texture smoothing. The user selects the desired technique by entering the corresponding number.

Specifically, the process when the program is running is as follows:

1. When the program is started, a list of technology options will be displayed and the user can select any number from 1 to 6 to specify the technology to be used.
2. For the first three techniques (Normal Cloning, Hybrid Cloning, Monochrome Conversion), the user needs to provide the path to the source and destination images. The program will load these images and perform the necessary checks to ensure that the images exist and can be loaded correctly.
3. For the next three techniques (color variation, lighting variation, and texture smoothing), the user needs to provide a source image and enter some parameters for adjusting the effect, such as RGB values, alpha and beta values, threshold value, etc.
4. The program simplifies the area selection in image processing through predefined polygonal areas, and sets the corresponding areas and points for the source image and the target image.
5. Then, the program will call the corresponding OpenCV function for processing according to the technique selected by the user, such as seamlessClone, colorChange, illuminationChange or textureFlattening. The processed result image will be saved to the specified file (such as cloned.png).
6. When processing is complete, the program prints a message telling the user that the resulting image has been saved to disk and displays the name of the saved file.

When the whole program ends, the processed image will not be displayed, but the result will be saved to a file. Users need to manually open these saved image files to view the processing effect. If the user enters the wrong option or image path, the program will output the corresponding error message and terminate the operation.

Run the use case:

|  |
| --- |
| ./example\_cpp\_cloning\_gui |

Running results (the red areas need to be entered by yourself):

|  |
| --- |
| Cloning Module  ---------------  Step 1:  -> In the source image, select the region of interest by left click mouse button. A Polygon ROI will be created by left clicking mouse button.  -> To set the Polygon ROI, click the right mouse button or use 'd' key  -> To reset the region selected, click the middle mouse button or use 'r' key.  Step 2:  -> In the destination image, select the point where you want to place the ROI in the image by left clicking mouse button.  -> To get the cloned result, click the right mouse button or use 'c' key.  -> To quit the program, use 'q' key.  Options:  1) Normal Cloning  2) Mixed Cloning  3) Monochrome Transfer  4) Local Color Change  5) Local Illumination Change  6) Texture Flattening  Press number 1-6 to choose from above techniques: 1  Enter Source Image: ./cloning/Normal\_Cloning/source1.png  Enter Destination Image: ./cloning/Normal\_Cloning/destination1.png |

## Test case 29: connected\_components

This program demonstrates how to use OpenCV to detect connected components of an image and extract and display these components by controlling the threshold with a slider. The program first converts the input image to grayscale, then binarizes it with a user-adjustable threshold, and finally identifies the connected regions in the image and assigns a random color to each connected region.

The main steps when the program is running are as follows:

1. The program takes the path to the input image from the command line argument and tries to read the image. If the image fails to load, the program outputs an error message and terminates.
2. The input image is converted to grayscale, which is the basis of the processing.
3. The main image processing is done in the on\_trackbar function. This function binarizes the grayscale image according to the current threshold and generates a binary image (i.e., a bw matrix) in which the areas with pixel values less than the threshold are black and the areas with pixel values greater than the threshold are white.
4. Next, the program uses the connectedComponents function to detect connected regions in the binary image, and each connected region is assigned a unique label.
5. The program assigns a random color to each detected connected region and stores the result in a color image where the background remains black and the connected regions are filled with random colors.
6. The final processed image will not be displayed, but directly saved as the file connected\_components.png. Users can view the saved image file after the program runs.

The program's end state is to output a message telling the user that the processed image has been saved and the name of the image file. If everything goes well, the program will exit successfully.

Run the use case:

|  |
| --- |
| ./example\_cpp\_connected\_components ../../samples/data/stuff.jpg |

Operation results:

|  |
| --- |
| This program demonstrates connected components and use of the trackbar  Usage: example\_cpp\_connected\_components [params] image  image (value:../../samples/data/stuff.jpg)  image for converting to a grayscale  The image is converted to grayscale and saved as output, another image has a trackbar  that controls thresholding and thereby the extracted contours which are drawn in color  Processing the image: ../../samples/data/stuff.jpg  Processed image saved as connected\_components.png |

## Test case 30: contours2

This program demonstrates how to use OpenCV's findContours and drawContours functions to detect and draw contours in an image. The program first generates a black and white image containing multiple ellipses and lines, then extracts contours from the image, and finally draws these contours according to the hierarchical parameters provided by the user and saves the result as an image file.

The main flow of the program is as follows:

1. **Help information output** : The program provides a help function that outputs instructions for use, explaining the purpose of the program and how to control the contour level through command line parameters.
2. **Command Line Parameter Parsing** : The program uses CommandLineParser to parse command line parameters. Users can specify the level of drawing contours through the @levels parameter (ranging from -3 to 3). If the parameter is not in the valid range, the program will output an error message and terminate.
3. **Generate test image** : The program generates a black and white image of size 500x500 pixels and draws 6 facial images in it, which are a combination of ellipses and lines to test the contour detection function.
4. **Contour detection** : The program uses the findContours function to perform contour detection on the generated image, extracts all contours in the image, and stores them in contours0. Subsequently, the program performs polygonal approximation on each detected contour to simplify the contour data.
5. **Draw contours** : The program calls the processContours function to draw contours according to the hierarchy level specified by the user. Specifically, the drawContours function determines which contours to draw based on the hierarchy parameters and saves the drawn contours as a color image.
6. **Save results** : The processed image will not be displayed, but will be saved directly as the file contours\_output.png. Users can view the saved image file after the program runs to observe the detected contours.

After the program is finished processing, it will output a message telling the user that the contour has been processed and saved to the specified file. The program will exit if everything is normal. If the user enters incorrect parameters or other errors occur, the program will output the corresponding error message and terminate the operation.

Run the use case:

|  |
| --- |
| ./example\_cpp\_contours2 |

Operation results:

|  |
| --- |
| Contours processed and saved to contours\_output.png |

## Test case 31: convexhull

This program demonstrates how to use OpenCV's convexHull function to compute and draw the convex hull of a set of points and save the result as an image file. The program generates a 500x500 pixel image, randomly generates a set of points, then computes the convex hull of these points and draws it on the image. Each resulting image is saved to a subdirectory, and the program loops to generate multiple image files.

The following is the detailed process flow:

1. **Help information output** : The program provides simple instructions through the help function, explaining the purpose of the program and how to run it.
2. **Command Line Parameter Parsing** : The program uses CommandLineParser to parse command line parameters. Users can view help information through the --help parameter. If the user requests help, the program will output help information and exit.
3. **Image and random number generator initialization** : The program initializes a 500x500 pixel image (black background) and uses OpenCV's random number generator RNG to generate random points.
4. **Create output directory** : The program tries to create a subdirectory named convexhull to store the generated image files. If the directory creation fails, the program will output an error message and terminate.
5. **Generate random points and calculate the convex hull** : The program performs the following steps in a loop:
   * Generate a random number of points (between 1 and 100) and generate these points in the central area of the image.
   * Use the convexHull function to compute the convex hull of these points, that is, the smallest convex polygon that contains all the points.
   * Clear the image and plot the randomly generated points on the image in red, then draw the calculated convex hull with a green line.
6. **Save the resulting image** : Each generated image will be saved as a file with the file name format frame\_X.png, where X is the frame counter. After each image is saved, the program will output the saved file name in the console.
7. **Loop control** : The program will generate and save multiple image files in a loop. In this example, assume that the program only generates and saves 10 frames, after which the loop ends and the program exits normally.

After the program ends, users can find saved image files in the convexhull directory, which show the effect of random points and their convex hull.

Run the use case:

|  |
| --- |
| ./example\_cpp\_convexhull |

Operation results:

|  |
| --- |
| Saved frame to convexhull/frame\_0.png  Saved frame to convexhull/frame\_1.png  Saved frame to convexhull/frame\_2.png  Saved frame to convexhull/frame\_3.png  Saved frame to convexhull/frame\_4.png  Saved frame to convexhull/frame\_5.png  Saved frame to convexhull/frame\_6.png  Saved frame to convexhull/frame\_7.png  Saved frame to convexhull/frame\_8.png  Saved frame to convexhull/frame\_9.png  Saved frame to convexhull/frame\_10.png |

## Test case 32: cout\_mat

This program demonstrates the serialization output capabilities of OpenCV's cv::Mat objects and applies them to format matrices and other OpenCV objects into a number of different styles (such as default, MATLAB, Python, NumPy, CSV, and C style) and output them to a text file. The program is intended to demonstrate how to use OpenCV's formatting tools to output the contents of matrices and other data structures.

The following is a description of the main flow and functions of the program:

1. **Help information output** : The program provides a help function to output instructions, explaining the purpose of the program and how to run it. If the user runs the program through the command line with the --help option, the program will display help information and exit.
2. **Open output file** : The program attempts to open a file called output.txt for writing. If the file cannot be opened, the program will output an error message and terminate.
3. **Create and print a matrix** :
   * **Create an identity matrix** : The program first creates a 4x4 double-precision floating-point identity matrix Mat::eye(4, 4, CV\_64F) and sets an element in the matrix to π (CV\_PI).
   * **Output matrix to file** : The identity matrix I will be output to a file in the default OpenCV format.
4. **Generate a random matrix and format it for output** :
   * The program creates a 10x3 three-channel 8-bit unsigned integer matrix r and fills it with random values generated by the randu function.
   * The matrix r will be output to the file in different formats, including default format, MATLAB format, Python format, NumPy format, CSV format, and C language style format. Each format demonstrates how to use the format function and different format options Formatter::FMT\_\* to format the output.
5. **Output points and vectors** :
   * The program creates a two-dimensional point Point2f and a three-dimensional point Point3f and outputs their contents to the file.
   * A simple floating-point vector v and a Point2f vector points containing multiple two-dimensional points are created and output. These points are generated according to a certain rule.
6. **Close the file and exit the program** :
   * After the program completes all output, it closes the output file output.txt and prints a message on the console to inform the user that the output has been written to the file.
   * The program exits normally.

The output of the entire program will be saved in the output.txt file, which the user can open to view the contents of matrices, points, and vectors output in different formats. This demonstrates the powerful formatting and output capabilities of OpenCV, especially when it needs to be interacted with other tools (such as MATLAB or Python).

Run the use case:

|  |
| --- |
| ./example\_cpp\_cout\_mat |

Operation results:

|  |
| --- |
| Output written to output.txt |

## Test case 33: create\_mask

This program demonstrates how to use OpenCV to create a binary mask image based on user input or a predefined set of points, and generate a result image with the mask applied. The program takes an input image and allows the user to define a polygonal area by mouse clicks, then generates a mask image corresponding to the area and a result image with the mask applied. Finally, both the mask image and the result image are saved to disk.

The following are the main functions and execution flow of the program:

1. **Help information output** : The program will output simple help information when it starts, explaining how to use mouse events to create mask shapes and explaining the different uses of the mouse buttons.
2. **Command Line Parameter Parsing** : The program uses CommandLineParser to parse the command line parameters. The user can specify the path of the input image through the @input parameter. By default, the program uses lena.jpg as the input image.
3. **Load input image** : The program attempts to load the specified input image. If the image fails to load, the program will output an error message and terminate.
4. **Create output directory** : The program creates a subdirectory named create\_mask to save the generated mask image and result image. If the directory creation fails, the program will output an error message and terminate the run.
5. **Mouse event handling** :
   * The program contains a mouseHandler function, which handles different mouse events, such as left click, right click and middle click. Users can define the vertices of the polygon on the image by clicking the left button, and complete the polygon drawing and generate the mask image and the result image by clicking the right button.
   * Middle-click to reset the defined vertices and clear the current mask and results.
6. **Mask creation and application** :
   * The program uses the fillPoly function in mouse event processing to generate a binary mask image based on the user-defined polygon. The mask area is white and the rest of the area is black.
   * The mask is applied to the original image using the bitwise\_and function, producing a result image that only shows the contents of the polygonal area.
7. **Save the image** :
   * The program saves the generated mask image and the result image with the mask applied to subdirectories with the file names mask.png and result.png respectively.
   * The program will output the save path and prompt the user that the mask and result image have been saved.
8. **End of program** :
   * The program exits normally after saving the image.

The core of the whole program is to create a mask through user input (mouse click) or a predefined set of points and save the generated mask and the image with the mask applied to disk. This approach is very useful in image processing tasks, especially when it is necessary to operate on specific areas of the image.

Run the use case:

|  |
| --- |
| ./example\_cpp\_create\_mask ../../samples/data/starry\_night.jpg |

Running results (saved in the data directory):

|  |
| --- |
| This program demonstrates using mouse events  Usage: example\_cpp\_create\_mask [params] input  input (value:../../samples/data/starry\_night.jpg)  input image  left mouse button - set a point to create mask shape  right mouse button - create mask from points  middle mouse button - reset |

## Test case 34: dbt\_face\_detection

This program demonstrates how to use OpenCV's CascadeClassifier and DetectionBasedTracker classes to detect faces in a video and save the results to a new video file. The program is first conditionally compiled to run on Linux, macOS, Android, and Windows platforms with Visual Studio 2013 or later. The CascadeDetectorAdapter class inherits from DetectionBasedTracker::IDetector and encapsulates the detection function of CascadeClassifier.

The program flow is as follows: open the input video file vtest.avi, load the LBP face classifier lbpcascade\_frontalface.xml, and then initialize the detector. Next, read the video frame by frame, convert each frame into a grayscale image, and use the detector to recognize the face. Draw a green rectangle on the recognized face and save the processed frame to the output.avi file. After processing, release all resources. If the program is run on an unsupported platform, it will output a corresponding prompt and exit.

Run the use case:

|  |
| --- |
| ./example\_cpp\_dbt\_face\_detection |

Operation results:

|  |
| --- |
| [ WARN:0@0.007] global samples.cpp:61 findFile cv::samples::findFile('samples/data/vtest.avi') => '/home/chenweijia/opencv/bin/cpp/../../samples/data/vtest.avi'  [ WARN:0@0.019] global samples.cpp:61 findFile cv::samples::findFile('data/lbpcascades/lbpcascade\_frontalface.xml') => '/home/chenweijia/opencv/bin/cpp/../../data/lbpcascades/lbpcascade\_frontalface.xml' |

## Test case thirty-five: delaunay2

This program shows how to use OpenCV for Delaunay triangulation and Voronoi diagram generation. The program randomly generates 200 points on a 600x600 pixel image, gradually constructs a Delaunay triangulation and the corresponding Voronoi diagram, and saves the result as an image file output.png.

The main steps are as follows:

1. **Initialization** : Create a 600x600 blank image and define a Subdiv2D object to handle the calculation of Delaunay triangulation and Voronoi diagram.
2. **Point-by-point insertion** : The program generates random points and inserts them into the Subdiv2D object one by one. After each insertion, the program updates the Delaunay triangle and displays the result.
3. **Draw Delaunay triangles** : After each insertion point, the program uses the draw\_subdiv function to draw all Delaunay triangles and show their edges.
4. **Draw the Voronoi diagram** : After all points are inserted, use the paint\_voronoi function to generate and draw the Voronoi diagram, and each Voronoi area is filled with a random color.
5. **Result saving** : The final generated image is saved as output.png.

This program is mainly used to demonstrate the dynamic construction process of Delaunay triangulation and Voronoi diagram, and is suitable for applications related to geometry processing and computational geometry.

Run the use case:

|  |
| --- |
| ./example\_cpp\_delaunay2 |

Operation results:

|  |
| --- |
| After running, the program will generate and save an image file named output.png, which contains the results of Delaunay triangulation and Voronoi diagram. There is no log after the program runs. |

## Test case thirty-six: demhist

This program demonstrates how to use OpenCV to adjust the brightness and contrast of an image and generate the corresponding histogram. The main function of the program is to adjust the image display effect by the brightness and contrast parameters set by the user, and calculate and display the grayscale histogram of the image. Finally, the adjusted image and histogram are saved as files.

The main steps are as follows:

1. **Initialization and parameter parsing** : The program parses the input image path through the command line parameters, loads the image and converts it to grayscale. If the image cannot be loaded, the program will output an error message and exit.
2. **Brightness and contrast adjustment** : The updateBrightnessContrast function adjusts the image according to the brightness and contrast values set by the user. The adjustment algorithm is based on the method of Werner D. Streidt, and the adjusted coefficient a and offset b are calculated and applied to the image.
3. **Histogram calculation** : The program uses the calcHist function to calculate the grayscale histogram of the adjusted image, and uses the normalize function to normalize it for easy display and saving.
4. **Result saving** : The adjusted image and the corresponding histogram are saved as result.png and histogram.png respectively.

The program finally outputs the processing results to the console and saves the image and histogram without displaying the graphical interface. This code shows how to combine the basic operations of image processing and histogram analysis.

Run the use case:

|  |
| --- |
| /example\_cpp\_demhist ../../samples/data/board.jpg |

Operation results:

|  |
| --- |
| Result image saved as: result.png  Histogram image saved as: histogram.png |

## Test case thirty-seven: detect\_blob

This program demonstrates how to use OpenCV's SimpleBlobDetector to detect and filter BLOBs (Binary Large Objects) in an image, and generate different detection results based on different parameter settings. BLOB detection is a method for detecting specific shapes or regions in an image, and is often used in shape analysis and image processing tasks.

The main functions and steps are as follows:

1. **Command line parsing and image loading** : The program takes the path of the input image from the command line (the default is lena.jpg) and loads the image. If the image cannot be loaded, the program will output an error message and exit.
2. **BLOB detection parameter configuration** : The program sets multiple SimpleBlobDetector::Params parameters, each of which corresponds to a BLOB detection strategy, such as detection based on area, roundness, inertia ratio, convexity, etc. These parameters are stored in the pBLOB vector and used to create different BLOB detectors.
3. **Color Palette Generation** : Generates a random color palette for drawing detected BLOBs in the image.
4. **BLOB detection and result saving** :
   * For each detector, the program detects BLOBs in the image using the detect function and draws the detected BLOBs on the image.
   * The resulting image for each detector is saved as a file whose file name contains a description of the detector's parameters.
5. **Result output** : The program saves each detection result as an image file and outputs the name of the saved file in the console.

The program uses different detection parameters to demonstrate how to filter and detect areas in an image based on specific shapes or attributes. It is suitable for application scenarios such as shape analysis and target detection.

Run the use case:

|  |
| --- |
| ./example\_cpp\_detect\_blob ../../samples/data/lena.jpg |

Operation results:

|  |
| --- |
| This program demonstrates how to use BLOB to detect and filter region  Usage:  ./example\_cpp\_detect\_blob <image1(lena.jpg as default)>  Press a key when image window is active to change descriptorResult image saved as: BLOB Area range [1 to 262144] AND Convexity range[0.9 to 1]\_result.png  Result image saved as: BLOB Area range [500 to 2900] AND Convexity range[0.9 to 1]\_result.png  Result image saved as: BLOB Circularity range [0.9 to 1e+37] AND Convexity range[0.9 to 1]\_result.png  Result image saved as: BLOB Convexity range[0.9 to 1] AND Inertia ratio range [0.1 to 0.2]\_result.png  Result image saved as: BLOB Convexity range[0.5 to 0.9]\_result.png  Result image saved as: BLOB Blob color 0 AND Convexity range[0.9 to 1]\_result.png |

## 测试用例三十九：detect\_mser

This program demonstrates how to use OpenCV's MSER (Maximally Stable Extremal Regions) algorithm to detect extreme regions in an image and save the result as an image file. MSER is a method for detecting regions with stable gray levels in an image, and is often used in applications such as text detection and object tracking.

The main steps are as follows:

1. **Command line parsing and image loading** : The program loads the input image via command line arguments. If no image path is provided, a synthetic test image is generated. The loaded image is blurred to reduce noise.
2. **MSER parameter configuration** : The program sets multiple MSERParams parameters to configure the MSER detector. The parameters include region size, grayscale change, region evolution, etc. These parameters are stored in the pMSER vector and used to create different MSER detectors.
3. **Detection and result saving** : The program uses MSER detectors with different configurations to detect regions in the image and marks the detected regions on the image. The marked regions are displayed in blue, and the number of detected regions and pixels is counted.
4. **Result output** : The detection result of each configuration is saved as an image file, and the statistical information of the detection result and the name of the saved file are output on the console.

The program demonstrates the process of extreme region detection through different MSER parameter configurations and saves the image results after each detection. This has wide applications in image analysis, target detection and other fields.

Run the use case:

|  |
| --- |
| ./example\_cpp\_detect\_mser ../../samples/data/detect\_blob.png |

Operation results:

|  |
| --- |
| This program demonstrates how to use MSER to detect extremal regions  Usage:  ./example\_cpp\_detect\_mser <image1(without parameter a synthetic image is used as default)>  Press esc key when image window is active to change descriptor parameter  Press 2, 8, 4, 6, +, -, or 5 keys in openGL windows to change view or use mouse  Number of MSER region: 228; Number of pixels in all MSER region: 469570  Result image saved as: MSERArea[100,5000] del. [10] var. [2] div. [0] pas. [1] RGb->evo. [200] are. [1] mar. [0] siz. [5]\_result.png  Number of MSER region: 257; Number of pixels in all MSER region: 501088  Result image saved as: MSERArea[100,5000] del. [10] var. [2] div. [0] pas. [0] RGb->evo. [200] are. [1] mar. [0] siz. [5]\_result.png  Number of MSER region: 50; Number of pixels in all MSER region: 77774  Result image saved as: MSERArea[100,5000] del. [100] var. [2] div. [0] pas. [0] RGb->evo. [200] are. [1] mar. [0] siz. [5]\_result.png |

## Test case 40: dft

This program demonstrates how to use OpenCV to perform a discrete Fourier transform (DFT) and display the spectrogram of an image. It is mainly used to analyze the frequency distribution in an image by converting the image from the spatial domain to the frequency domain through Fourier transform.

The main steps of the procedure are as follows:

1. **Load image** : The program loads the input grayscale image (the default is lena.jpg) through the command line parameters. If the image cannot be loaded, the program will output an error message and exit.
2. **Optimize DFT size** : To improve the efficiency of DFT calculation, the program uses the getOptimalDFTSize function to calculate the optimal DFT size, expands the image to this size, and uses the copyMakeBorder function to fill the image border to make it suitable for DFT processing.
3. **Fourier Transform** : The program creates a complex image matrix containing real and imaginary parts and performs a Fourier transform on the matrix, producing a frequency domain representation.
4. **Calculate the magnitude spectrum** : Calculate the magnitude of the spectrum by separating the real and imaginary parts and take its logarithm to better display the low-frequency components.
5. **Adjust spectrum image** : To observe the spectrum more intuitively, the program rearranges the quadrants of the spectrum graph so that the low-frequency component is located in the center of the image.
6. **Normalization and Saving** : The program normalizes the spectrogram so that its values are between 0 and 1, and saves the final spectrogram as the "spectrum\_magnitude.png" file.

The final result image shows the frequency distribution of the image, and the program is suitable for image processing tasks such as spectrum analysis and filter design.

Run the use case:

|  |
| --- |
| ./example\_cpp\_dft ../../samples/data/lena.jpg |

Operation results:

|  |
| --- |
| This program demonstrates the use of the discrete Fourier transform (dft)  The dft of an image is taken and its power spectrum is displayed.  Usage:  ./example\_cpp\_dft [image\_name -- default lena.jpg]  Result image saved as: spectrum\_magnitude.png |

## Test case 41: digits\_lenet

This program demonstrates how to use OpenCV's deep learning module (DNN) for image classification. It captures images from a camera, video file, or image file and uses a pre-trained neural network to classify the objects in the image. This program is mainly used to recognize and label numbers or other objects in images.

**Main functions and steps:**

1. **Command line parsing and network loading** :
   * The input source (image, video or camera) and the path to the neural network model are parsed through command line arguments.
   * Load the pre-trained Caffe model (modelBin) and model definition file (modelTxt), and build a neural network.
2. **Video stream processing** :
   * Captures image frames from the specified input source. If no input file is specified, the camera is used by default.
   * Each frame is processed in a loop, performing preprocessing, connected component detection, and object recognition.
3. **Image preprocessing** :
   * Convert the image to grayscale and perform Gaussian blur and adaptive thresholding.
   * Morphological operations (dilation) are used to enhance the edges of objects in the image to facilitate subsequent connected component detection.
4. **Connected component detection** :
   * Use connectedComponentsWithStats to detect connected components in an image and filter out potential object regions based on properties such as area.
5. **Object Classification** :
   * A deep learning network is used to classify each screened area to obtain the most likely category and probability.
   * The low-probability results are filtered out by setting the confidence threshold (thr) and the classification results are marked in the original image.
6. **Save the results** :
   * Draw the detected object bounding boxes and classification results in the image and save the processed image.
7. **Performance Measurements** :
   * The FPS (frames per second) of processed frames is calculated and displayed to evaluate the real-time performance of the model.

Finally, the processed image will be saved to the specified directory, and the save path and processing performance information will be output to the console. This program is suitable for real-time image classification tasks, such as digit recognition or object detection.

Run the use case:

|  |
| --- |
| ./example\_cpp\_digits\_lenet --input=../../samples/data/vtest.avi --modelTxt=lenet.prototxt --modelBin=lenet.caffemodel |

Operation results:

|  |
| --- |
| Processed image saved at: digits\_lenet/processed\_image.jpg |

## Test case 42: digits\_svm

This program demonstrates how to use OpenCV's SVM (Support Vector Machine) and K Nearest Neighbor (KNN) algorithms for handwritten digit recognition. The program loads data from the image file digits.png containing handwritten digits, splits it into individual digit images of 20x20 pixels, and assigns a corresponding label (0-9) to each digit. After the data is loaded, each digit image is de-skewed to reduce the effect of writing tilt.

Next, the program uses the HOG (Histogram of Oriented Gradients) feature extraction method to convert each digit image into a feature vector, which is normalized and converted to a Hellinger kernel. The dataset is then randomly shuffled and divided into a training set (90%) and a test set (10%) for training and evaluating KNN and SVM models.

The program predicts the test set and outputs the confusion matrix and error rate. The test results are saved as image files (such as KNearest\_test.png and SVM\_test.png). Finally, the trained SVM model is saved as digits\_svm.yml for later use. This program is suitable for beginners in image processing and machine learning, and shows how to perform handwritten digit recognition.

Run the use case:

|  |
| --- |
| ./example\_cpp\_digits\_svm dis\_opticalflow ./example\_cpp\_dis\_opticalflow --video=../../samples/data/vtest.avi |

Operation results:

|  |
| --- |
| SVM and KNearest digit recognition.  Sample loads a dataset of handwritten digits from 'digits.png'.  Then it trains a SVM and KNearest classifiers on it and evaluates  their accuracy.  Following preprocessing is applied to the dataset:  - Moment-based image deskew (see deskew())  - Digit images are split into 4 10x10 cells and 16-bin  histogram of oriented gradients is computed for each  cell  - Transform histograms to space with Hellinger metric (see [1] (RootSIFT))  [1] R. Arandjelovic, A. Zisserman  "Three things everyone should know to improve object retrieval"  http://www.robots.ox.ac.uk/~vgg/publications/2012/Arandjelovic12/arandjelovic12.pdf |

## Test case 43: dis\_opticalflow

This program demonstrates how to use OpenCV to calculate dense optical flow for analyzing motion in videos. The program uses the DISOpticalFlow algorithm to read data from the video frame by frame, calculate the optical flow between each frame, and visualize the optical flow in a color-coded way.

**Main functions and steps:**

1. **Video read** :
   * The program specifies the input video file (default is vtest.avi) through command line parameters and uses VideoCapture to open the video. If the video file cannot be opened, the program will output an error message and exit.
2. **Optical flow calculation** :
   * The program reads the video frame by frame and converts each frame into a grayscale image.
   * Between consecutive frames, the DISOpticalFlow algorithm is used to calculate dense optical flow, which describes the direction and speed of movement of each pixel in the image.
   * By separating the horizontal and vertical components of the optical flow, the magnitude and angle of motion are calculated.
3. **Optical flow visualization** :
   * Convert the magnitude and angle of optical flow to HSV color space, where the angle is mapped to hue, the magnitude is mapped to saturation, and the brightness is fixed.
   * Convert HSV images to BGR color space for easy display and storage.
4. **Result processing** :
   * Although the image display and key-waiting sections are commented out, the program can be used to batch process videos and analyze motion patterns in videos.

The program is suitable for video analysis tasks, especially in motion detection and tracking applications.

Run the use case:

|  |
| --- |
| ./example\_cpp\_dis\_opticalflow --video=../../samples/data/vtest.avi |

Operation results:

|  |
| --- |
| Processing completed. |

## Test case 44: distrans

This program demonstrates how to use OpenCV's distance transform function to process binary images. Distance transform is a method that calculates the distance from each pixel in an image to the nearest edge pixel, and is often used in applications such as shape analysis, edge detection, and image segmentation. The program can also generate a Voronoi diagram.

**Main functions and steps:**

1. **Loading an image** :
   * The program loads the input image passed as a command line argument and converts it to a grayscale image. If the image cannot be loaded, the program outputs an error message and exits.
2. **Distance Transform** :
   * The program calculates the distance transform of a grayscale image based on the distance metric (L1, L2, C/Inf) and mask size (3x3 or 5x5) specified by the user.
   * When Voronoi mode is on, the program also computes the Voronoi diagram, which divides the image into distance-based regions.
3. **Result processing and visualization** :
   * The result of the distance transform is scaled and smoothed to enhance the visual effect.
   * In the Voronoi pattern, the program uses different colors to represent different areas in the image.
   * The final processing result is saved as distance\_map.png, and the saved information is output.
4. **Functional description** :
   * The program supports multiple distance metrics and mask sizes, and provides the option of generating Voronoi diagrams. These parameters can be dynamically switched by modifying the code or adding a corresponding user interface.

This program is suitable for shape analysis and edge detection tasks in image processing, especially in application scenarios where distance information is required.

Run the use case:

|  |
| --- |
| ./example\_cpp\_distrans --image=../../samples/data/stuff.jpg |

Operation results:

|  |
| --- |
| Program to demonstrate the use of the distance transform function between edge images.  Usage:  ./example\_cpp\_distrans [image\_name -- default image is stuff.jpg]  Hot keys:  ESC - quit the program  C - use C/Inf metric  L1 - use L1 metric  L2 - use L2 metric  3 - use 3x3 mask  5 - use 5x5 mask  0 - use precise distance transform  v - switch to Voronoi diagram mode  p - switch to pixel-based Voronoi diagram mode  SPACE - loop through all the modes  Result image saved as: distance\_map.png |

## 测试用例四十五：drawing

This program demonstrates how to use OpenCV's drawing and text output functions. By randomly generating geometric shapes and text, it shows the drawing functions of lines, rectangles, ellipses, circles, filled polygons, etc., and also shows the text rendering function. The entire image is drawn by using the RNG random number generator to create random colors, positions and sizes.

**Key features and steps:**

1. **Random graphics drawing** :
   * The program draws various random graphics on a blank image, including lines, arrows, rectangles, ellipses, circles, polygons, etc. The color, position and size of each graphic are randomly generated.
2. **Text drawing** :
   * Use the putText function to randomly place text on the image, demonstrating control over the font, size, and color.
3. **Image processing and saving** :
   * The program displays the dynamic effect by processing the entire image (such as color fading) and drawing the text again. The final processed image is saved as drawing\_demo.png through the imwrite function.
4. **Program Structure** :
   * The program uses multiple loops to achieve batch drawing of different types of graphics, and generates random colors through the defined randomColor function. After the graphics are drawn, the program saves the result as a PNG image and outputs the save path.

This example is suitable for learning and demonstrating the basic drawing functions of OpenCV, helping to understand how to draw graphics and text on images, and how to use OpenCV's random number generation and image processing functions.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Operation results:

|  |
| --- |
| This program demonstrates OpenCV drawing and text output functions.  Usage:  ./example\_cpp\_drawing  Result image saved as: drawing\_demo.png |

## Test case 46: edge

This program demonstrates Canny edge detection using OpenCV, including the default Canny edge detection and Canny edge detection using the Scharr filter. The program reads an input image, performs edge detection operations, and saves the results as an image file.

**Key steps:**

1. **Read and preprocess the image** :
   * The program first reads the specified image file and converts it into a grayscale image for subsequent edge detection operations.
2. **Canny edge detection** :
   * The program uses the Canny algorithm to detect edges in images. The first part uses the default Canny parameters for processing, and the results are saved in canny\_default.png.
3. **Scharr filter and Canny edge detection** :
   * In the second part, the program first calculates the gradient of the image through the Scharr filter, and then uses these gradient results to perform Canny edge detection. The result is saved in canny\_scharr.png.
4. **Save the results** :
   * The edge detection results of both methods are saved as image files in PNG format, and the file names are output for users to view.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Operation results:

|  |
| --- |
| This sample demonstrates Canny edge detection  Call:  ./example\_cpp\_edge [image\_name -- Default is fruits.jpg]  Result image saved as: canny\_default.png  Result image saved as: canny\_scharr.png |

## Test case 47: ela

This code implements an Error Level Analysis (ELA) function for a JPEG image. ELA is a method to detect whether an image has been digitally modified by analyzing the changes in the image at different compression levels. The principle is that the various parts of a JPEG image should be at the same or similar compression level. If the error level of a part is significantly different, it may indicate that the part has been edited.

When run, the program first reads the input JPEG image, then recompresses the image, setting compression quality parameters. Next, it calculates the difference between the original image and the recompressed image, which reflects the error level of the image. The program visualizes this difference using a scaling value, generates an ELA result image, and saves it to disk. At the same time, the recompressed image is also saved.

The main function of the program is to help users identify image modification traces by outputting ELA result images when there are different compression errors in the image. At the end of the program, two image files will be generated and saved: one is the recompressed image, and the other is the result image after ELA analysis.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Operation results:

|  |
| --- |
| Jpeg Recompression Example:  Usage: example\_cpp\_ela [params]  -i, --input (value:../../samples/data/ela\_modified.jpg)  Input image to calculate ELA algorithm.  Compressed image saved as: compressed\_image.jpg  ELA result image saved as: ela\_result.png |

## Test case 48: em

This code implements a clustering example based on the expectation maximization (EM) algorithm and visualizes the clustering results as an image. The EM algorithm is a statistical model method commonly used for data clustering. It can estimate the parameters of the Gaussian mixture model in an iterative manner to divide the data into multiple categories.

When the program is running, a set of two-dimensional data samples are first generated. These samples are divided into 4 classes, and the data of each class is distributed in different areas of the image. The coordinates of each sample are generated by normal distribution, where the mean is set according to the location of the class, and the standard deviation is small, so that the sample points are gathered near the center of their respective classes.

Next, the program creates and trains an EM model to classify the data samples into four clusters. After the model is trained, the program classifies each image pixel and colors the pixel according to the category predicted by the model. Finally, these pixels are marked with different colors to indicate the different cluster areas they belong to. The training data samples are also drawn on the image and marked with the corresponding colors.

When the program ends, the generated clustering result image will be saved to disk and the save path of the image file will be displayed. The final image shows the distribution of sample points and the clustering effect of the EM algorithm.

Run the use case:

|  |
| --- |
| /example\_cpp\_em |

Operation results:

|  |
| --- |
| Result image saved as: em\_clustering\_result.png |

## Test case 49: epipolar\_lines

This code implements a basic stereo vision calculation process, by extracting feature points from two images, calculating their fundamental matrices, and drawing the corresponding epipolar lines. Epipolar lines are a very important concept in stereo vision, which are used to constrain the possible positions of matching feature points in another image.

When running, the program first extracts feature points from the two input images and calculates their descriptors. Then, it uses the FLANN matcher to find matching pairs of feature points between the two images and selects good matching points through the Lowe's ratio test. After that, the program uses the RANSAC algorithm to calculate the basic matrix between the two images, which describes the geometric relationship between the two images.

After calculating the fundamental matrix, the program draws the epipolar lines corresponding to the matching feature points and marks these feature points in the image. The program randomly selects a subset of feature points for drawing the epipolar lines and assigns them random colors. Finally, the program stitches the two images together and resizes the images to fit the specified dimensions. The processed image is saved as epipolar\_lines.png, showing the results of marking the feature points and epipolar lines.

The output of the program is an epipolar image saved to disk, which helps users intuitively observe the geometric relationship and matching effect in stereo matching.

Run the use case:

|  |
| --- |
| ./example\_cpp\_epipolar\_lines ../../samples/data/left01.jpg ../../samples/data/right02.jpg |

Operation results:

|  |
| --- |
| Number of points 168  RANSAC fundamental matrix time 6658  Mean distance from tentative inliers to epipolar lines 568.773 number of inliers 109  Result image saved as: epipolar\_lines.png |

## Test case fifty: essential\_mat\_reconstr

This code implements the function of reconstructing 3D points and extracting planes from two images, and finally visualizes the results on the image. The program goes through a series of calculation steps, first estimating the basic geometric relationship (essential matrix) between the two images, then using these relationships to reconstruct the 3D points in the scene, and using the RANSAC algorithm to fit several planes among these 3D points.

Specifically, the main process of the program is as follows:

1. **Image reading and feature detection** : Read two images from the specified file and extract feature points and their descriptors using the SIFT algorithm. Then, use the FLANN matcher to match feature points between the two images.
2. **Feature matching and essential matrix calculation** : The Lowe ratio test is used to select good feature point matching pairs, and then the RANSAC algorithm is used to estimate the essential matrix between the two images. During the calculation process, the program also checks for inliers, which are feature point pairs that conform to the geometric model.
3. **Triangulation and 3D point reconstruction** : Using the estimated essential matrix, the feature point matching pairs are triangulated to generate a set of 3D points that represent the object points in space.
4. **Plane fitting** : Use the RANSAC algorithm to fit multiple planes in the 3D point set and mark each 3D point to indicate the plane it belongs to.
5. **Result visualization** : All plane points and inliers are plotted onto the original image, using different colors to represent different planes. Finally, the two images are stitched together and saved as an image file planes.png.

At the end of the program, the generated plane image will be output, showing the feature points in the original image and the plane to which they belong, which can be used to observe the geometric relationship between images and the reconstruction effect of the scene structure.

Run the use case:

|  |
| --- |
| ./example\_cpp\_essential\_mat\_reconstr ../../samples/data/essential\_mat\_data.txt ../../samples/data/ |

Operation results:

|  |
| --- |
| RANSAC essential matrix time 18575mcs.  Number of inliers 220  Mean error to epipolar lines 0.268088  Number of object points 220  Result image saved as: planes.png |

## Test case 51: facedetect

This code implements a face and eye detection program based on OpenCV. The program uses Haar or LBP features to detect faces and eyes in images, and uses the CascadeClassifier class to load a pre-trained classifier model to first detect faces and then further detect eyes within the detected face area. The program supports capturing images from a camera in real time or reading images from a file for processing.

The program first parses the input parameters from the command line, such as the classifier path, image scaling, and whether to try image flipping. Then, the face and eye detectors are initialized according to the input parameters and the input image or video stream is read. If the program detects a face, a circle or rectangular box is drawn on the original image using the specified color to mark the detected face and eyes.

In actual operation, the program grayscales and histogram equalizes each frame, and then applies the detectMultiScale method to detect faces. If the flip option is enabled, the program will also detect mirrored images to improve the robustness of the detection. The detected faces and eyes are marked with different colors, and the final result image is saved as result.png.

The design of the entire process enables the program to flexibly process real-time video and static images, and by saving the processing results, users can easily view the detection effects.

Run the use case:

|  |
| --- |
| /example\_cpp\_facedetect --cascade="../../data/haarcascades/haarcascade\_frontalface\_alt.xml" --nested-cascade="../../data/haarcascades/haarcascade\_eye\_tree\_eyeglasses.xml" --scale=1.3 ../../samples/data/lena.jpg |

Operation results:

|  |
| --- |
| Detecting face(s) in ../../samples/data/lena.jpg  detection time = 53.6192 ms  Result image saved as: result.png |

## Test case 52: facial\_features

This code implements a program for detecting facial landmarks by using OpenCV's Haar cascade classifier to detect faces, eyes, noses, and mouths. The program first detects faces from the input image, and then further detects the locations of eyes, noses, and mouths within the detected facial regions. Users can specify the classifier path to use via command line parameters.

The main steps of the procedure are as follows:

1. **Image loading and parameter parsing** : The program obtains the path of the input image and the required Haar cascade classifier file path from the command line, including classifiers for face, eyes, nose, and mouth. If necessary parameters are missing, the program will output a help message and exit.
2. **Face detection** : The program first uses the provided face classifier to detect faces in the image and marks the detected face area with a rectangular frame.
3. **Feature point detection** : In each detected face area, the program further uses the corresponding classifier to detect eyes, nose and mouth. The center positions of the eyes and nose are marked with circles, and the mouth area is marked with a rectangular box.
4. **Conditional judgment of feature points** : The program ensures that the mouth is below the nose and adjusts the marking method of the detection results based on this condition.
5. **Result output** : The final processed image will be saved as result.png. Users can view the saved image to observe the detection effect.

This program is a basic facial feature detection example that shows how to use OpenCV's Haar cascade classifier for multi-level feature detection. By adjusting the classifier parameters and model, the program can provide reliable facial feature detection results under different conditions.

Run the use case:

|  |
| --- |
| ./example\_cpp\_facial\_features ../../samples/data/lena.jpg ../../data/haarcascades/haarcascade\_frontalface\_alt.xml --eyes=../../data/haarcascades/haarcascade\_eye.xml --mouth=../../data/haarcascades/haarcascade\_smile.xml |

Operation results:

|  |
| --- |
| Result image saved as: result.png |

## Test case fifty-three: falsecolor

This code implements a program for detecting facial landmarks by using OpenCV's Haar cascade classifier to detect faces, eyes, noses, and mouths. The program first detects faces from the input image, and then further detects the locations of eyes, noses, and mouths within the detected facial regions. Users can specify the classifier path to use via command line parameters.

The main steps of the procedure are as follows:

1. **Image loading and parameter parsing** : The program obtains the path of the input image and the required Haar cascade classifier file path from the command line, including classifiers for face, eyes, nose, and mouth. If necessary parameters are missing, the program will output a help message and exit.
2. **Face detection** : The program first uses the provided face classifier to detect faces in the image and marks the detected face area with a rectangular frame.
3. **Feature point detection** : In each detected face area, the program further uses the corresponding classifier to detect eyes, nose and mouth. The center positions of the eyes and nose are marked with circles, and the mouth area is marked with a rectangular box.
4. **Conditional judgment of feature points** : The program ensures that the mouth is below the nose and adjusts the marking method of the detection results based on this condition.
5. **Result output** : The final processed image will be saved as result.png. Users can view the saved image to observe the detection effect.

This program is a basic facial feature detection example that shows how to use OpenCV's Haar cascade classifier for multi-level feature detection. By adjusting the classifier parameters and model, the program can provide reliable facial feature detection results under different conditions.

Run the use case:

|  |
| --- |
| ./example\_cpp\_falsecolor ../../samples/data/lena.jpg |

Operation results:

|  |
| --- |
| This program demonstrates the use of applyColorMap function.  Gray image saved as: gray\_image.png  Result image saved as: colormap\_result.png  Processing completed. Check the saved images. |

## Test case 54: fback

This code shows how to use OpenCV to implement the calculation of dense optical flow, specifically through Gunnar Farneback's algorithm. Optical flow algorithms are used to detect and track pixel motion between video frames. The program can get video frames from a camera or video file, calculate the optical flow, and visualize the results on an image.

The main flow of the program is as follows:

1. **Parameter parsing and video capture initialization** : The program first parses the command line parameters to determine whether the input source is a camera or a video file. Then it uses the VideoCapture object to open the video source. If the video file path is not provided, the program tries to open the default camera.
2. **Video frame processing and optical flow calculation** : The program reads the video frames in a loop and converts them into grayscale images. If there is already a grayscale image of the previous frame, the program will calculate the optical flow between the two frames. The optical flow calculation uses the calcOpticalFlowFarneback function, which implements Farneback's dense optical flow algorithm.
3. **Visualization of optical flow** : The calculated optical flow results are visualized by the drawOptFlowMap function. This function draws the optical flow vector on the image, showing the movement of each pixel as an arrow pointing from the origin to the target position in green.
4. **Result saving** : The program saves the visualized optical flow image as optical\_flow\_result.png and then exits the loop. This example only processes and saves the optical flow result of the first frame as a demonstration.

Finally, the program generates and saves an image showing the motion of each pixel in the first frame of video. This demonstration can help to understand the concept of dense optical flow and show how to implement this calculation using OpenCV.

Run the use case:

|  |
| --- |
| ./example\_cpp\_fback ../../samples/data/vtest.avi |

Operation results:

|  |
| --- |
| Result image saved as: optical\_flow\_result.png |

## Test case 55: ffilldemo

This code demonstrates how to use OpenCV's floodFill() function to perform an area filling operation. The program allows the user to trigger the flood fill algorithm by clicking the mouse on the image, and the filling range is determined by color difference and connectivity. The user can switch between color and grayscale mode, use or not use a mask, and select different fill modes (simple fill, fixed range fill, or floating range fill).

When the program is running, the user selects the seed point by clicking the image with the mouse. The program calculates the area to be filled according to the selected mode and fills it with random colors. The user can also specify the image to be processed through command line parameters. The processed image is saved as output.jpg.

This example shows how to flexibly use the OpenCV floodFill() function and implement real-time image processing through user interaction.

Run the use case:

|  |
| --- |
| ./example\_cpp\_ffilldemo ../../samples/data/fruits.jpg |

Operation results:

|  |
| --- |
| This program demonstrates the floodFill() function  Call:  ./example\_cpp\_ffilldemo [image\_name -- Default: fruits.jpg]  Hot keys:  ESC - quit the program  c - switch color/grayscale mode  m - switch mask mode  r - restore the original image  s - use null-range floodfill  f - use gradient floodfill with fixed (absolute) range  g - use gradient floodfill with floating (relative) range  4 - use 4-connectivity mode  8 - use 8-connectivity mode  Processed image saved as: output.jpg |

## 测试用例五十六：filestorage

This code demonstrates how to use OpenCV's FileStorage class for serialization and deserialization operations. The program shows how to save custom structures, matrices, and lists of image file names to files in YAML or XML format and read data from these files.

The main process is as follows:

1. **Serialize data** : The program first creates a file containing a list of image file names, a matrix, and a custom structure MyData, and writes it to the specified output file. The custom structure is serialized through the overloaded write() method.
2. **Deserialize data** : The program then opens and reads the data from the file. It checks the file structure, verifies that the image list is of type sequence, reads the matrices R and T and the structure MyData, and displays the contents.
3. **String Manipulation** : The program also shows how to read and write serialized data from strings. This includes saving data as a YAML formatted string and recovering structured data from it.

Through this example, users can learn how to use OpenCV's FileStorage function to efficiently save and read complex data structures between different file formats.

Run the use case:

|  |
| --- |
| ./example\_cpp\_filestorage output .yml.gz |

Operation results:

|  |
| --- |
| writing images  image1.jpg myfi.png baboon.jpg  writing mats  R = [1, 0, 0;  0, 1, 0;  0, 0, 1]  T = [0;0;0]  writing MyData struct  {id = mydata1234, X = 3.14159, A = 97}  reading images  image1.jpg  myfi.png  baboon.jpg  reading R and T  R = [1, 0, 0;  0, 1, 0;  0, 0, 1]  T = [0;0; 0]  read mdata  { id = mydata1234, X = 3.14159, A = 97}  attempting to read mdata\_b  read mdata\_b  { id = , X = 0, A = 0}  Try opening output.yml.gz to see the serialized data.  Read data from string  attempting to read mdata from string  read mdata  { id = mydata1234, X = 3.14159, A = 97}  Write data to string  writing MyData struct  { id = mydata1234, X = 3.14159, A = 97}  Created string:  %YAML:1.0  ---  mdata:  A: 97  X: 3.1415926535897931  id: mydata1234  Serialized data saved to file: output.yml.gz |

## Test case fifty-seven: fitellipse

This code demonstrates three methods of ellipse fitting using OpenCV, including the OpenCV original method, the AMS method, and the direct least squares method. The program first finds the contours from the input image and fits the ellipse to these contour points. The results of the three methods are plotted on the image in different colors.

The main steps of the program are as follows:

1. **Initialization and image reading** : The program accepts the image path as input, loads the image and initializes the canvas, preparing for ellipse fitting and drawing.
2. **Contour detection** : The image is converted into a binary image through threshold processing, and the findContours function is used to detect contours in the image.
3. **Ellipse fitting and drawing** : Three methods are used to fit an ellipse for each contour. fitEllipse, fitEllipseAMS and fitEllipseDirect correspond to three fitting algorithms. The results are drawn on the image with lines of different colors, and white points represent the original contour points.
4. **Result Saving** : Finally, the processed image is saved as result.jpg, showing the fitted ellipse and the original data points.

This program can be used to compare the effects of different ellipse fitting algorithms and visually see how they perform on complex contours.

Run the use case:

|  |
| --- |
| ./example\_cpp\_filestorage ../../samples/data/ellipses.jpg |

Operation results:

|  |
| --- |
| sudo perf stat -e duration\_time,task-clock,cycles,instructions,cache-references,cache-misses,branches,branch-misses,L1-dcache-loads,L1-dcache-load-misses,LLC-loads,LLC-load-misses ./example\_cpp\_filestorage ../../samples/data/ellipses.jpg  writing images  image1.jpg myfi.png baboon.jpg  writing mats  R = [1, 0, 0;  0, 1, 0;  0, 0, 1]  T = [0; 0; 0]  writing MyData struct  { id = mydata1234, X = 3.14159, A = 97}  reading images  image1.jpg  myfi.png  baboon.jpg  reading R and T  R = [1, 0, 0;  0, 1, 0;  0, 0, 1]  T = [0;  0;  0]  read mdata  { id = mydata1234, X = 3.14159, A = 97}  attempting to read mdata\_b  read mdata\_b  { id = , X = 0, A = 0}  Try opening ../../samples/data/ellipses.jpg to see the serialized data.  Read data from string  attempting to read mdata from string  read mdata  { id = mydata1234, X = 3.14159, A = 97}  Write data to string  writing MyData struct  { id = mydata1234, X = 3.14159, A = 97}  Created string:  %YAML:1.0  ---  mdata:  A: 97  X: 3.1415926535897931  id: mydata1234 |

## Test case 58: flann\_search\_dataset

Run the use case:

|  |
| --- |
| ./example\_cpp\_flann\_search\_dataset --dataset=../../samples/data --image=../../samples/data/lena.jpg |

Operation results:

|  |
| --- |
| Processed image saved as: good\_matches.jpg |

## Test case fifty-nine: grabcut

This code shows how to use OpenCV's GrabCut algorithm for image segmentation. GrabCut is an interactive segmentation algorithm used to extract foreground objects from an image.

The main process is as follows:

1. **Initialization and image loading** : The program takes the file name of the input image from the command line argument and loads the image. If the image cannot be loaded, the program will exit.
2. **Application of GrabCut algorithm** : GCApplication class is responsible for image segmentation. It initializes the image and mask and applies GrabCut algorithm in the specified rectangular area to segment the foreground object. The program initializes the segmentation in the rectangle and optimizes the segmentation result through iteration.
3. **Result saving** : The processed image is saved in the grabcut directory as result.png. The saved image shows the segmented foreground object and the segmentation boundary is marked in the rectangular area.

This sample program shows how to simply use the GrabCut algorithm to extract foreground objects from an image and save the resulting processed image.

Run the use case:

|  |
| --- |
| ./example\_cpp\_grabcut --input=../../samples/data/messi5.jpg |

Operation results:

|  |
| --- |
| This program demonstrates GrabCut segmentation -- select an object in a region  and then grabcut will attempt to segment it out.  Call:  ./example\_cpp\_grabcut <image\_name>  Select a rectangular area around the object you want to segment  Hot keys:  ESC - quit the program  r - restore the original image  n - next iteration  left mouse button - set rectangle  CTRL+left mouse button - set GC\_BGD pixels  SHIFT+left mouse button - set GC\_FGD pixels  CTRL+right mouse button - set GC\_PR\_BGD pixels  SHIFT+right mouse button - set GC\_PR\_FGD pixels  Processed image saved at: grabcut/result.png |

## Test case sixty: image\_alignment

This code demonstrates how to use OpenCV's ECC (Enhanced Correlation Coefficient) algorithm for image alignment. The ECC algorithm is used to align two images accurately and is often used for image registration.

The main process is as follows:

1. **Parameter parsing and image loading** : The program obtains the input image, template image and initial transformation matrix from the command line. If no template image is provided, the program will randomly transform the input image to generate a template.
2. **Select transformation model** : Users can select four transformation models (translation, Euclidean transformation, affine transformation and homography transformation) for image alignment. The program initializes or reads the initial transformation matrix from a file.
3. **Perform ECC image alignment** : Use the findTransformECC function to align the input images according to the selected transformation model. The number of iterations and convergence criteria are specified by the user.
4. **Result saving** : The program saves the final transformation matrix and aligned image, and outputs the processing time and aligned result image.

This example shows how to use the ECC algorithm to achieve precise alignment of images and verifies the effect by saving the result.

Run the use case:

|  |
| --- |
| ./example\_cpp\_image\_alignment ../../samples/data/fruits.jpg |

Operation results:

|  |
| --- |
| ECC demo  Usage: example\_cpp\_image\_alignment [params] inputImage templateImage inputWarp  -e, --epsilon (value:0.0001)  ECC's convergence epsilon  -h, --help  print help message  -m, --motionType (value:affine)  type of motion (translation, euclidean, affine, homography)  -n, --numOfIter (value:50)  ECC's iterations  -o, --outputWarp (value:outWarp.ecc)  output warp (matrix) filename  -v, --verbose (value:1)  display initial and final images  -w, --warpedImfile (value:warpedECC.png)  warped input image  inputImage (value:../../samples/data/fruits.jpg)  input image filename  templateImage  template image filename (optional)  inputWarp  input warp (matrix) filename (optional)  This file demonstrates the use of the ECC image alignment algorithm. When one image is given, the template image is artificially formed by a random warp. When both images are given, the initialization of the warp by command line parsing is possible. If inputWarp is missing, the identity transformation initializes the algorithm.  Usage example (one image):  ./example\_cpp\_image\_alignment fruits.jpg -o=outWarp.ecc -m=euclidean -e=1e-6 -N=70 -v=1  Usage example (two images with initialization):  ./example\_cpp\_image\_alignment yourInput.png yourTemplate.png yourInitialWarp.ecc -o=outWarp.ecc -m=homography -e=1e-6 -N=70 -v=1 -w=yourFinalImage.png  ->Performance Warning: Identity warp ideally assumes images of similar size. If the deformation is strong, the identity warp may not be a good initialization.  Alignment time (affine transformation): 0.282566 sec  The final warp has been saved in the file: outWarp.ecc  The warped image has been saved in the file: warpedECC.png  Press any key to exit the demo (you might need to click on the images before). |

## Test case 61: imagelist\_creator

This code demonstrates how to use OpenCV's FileStorage function to save a list of files (such as image files) passed in as command line arguments to a file in YAML or XML format.

The main process is as follows:

1. **Command line parameter parsing** : The program obtains the name of the output file through the command line. If no output file is specified, the program will display a help message and exit.
2. **Check if the output file is an image** : To prevent accidental overwriting of image files, the program first attempts to read the output file name as an image. If the read succeeds (indicating that the file is an image), the program stops and prompts the user to re-specify the output file.
3. **File List Saving** : The program uses OpenCV's FileStorage function to save the file list specified in the command line parameter to a file in YAML or XML format.
4. **Result output** : The generated file will contain a list called images, listing all the input file paths.

This example shows how to use OpenCV's FileStorage functionality to conveniently save a list of files and ensure that existing image files are not accidentally overwritten.

Run the use case:

|  |
| --- |
| ./example\_cpp\_imagelist\_creator imagelist.yaml ../../samples/data/aloeGT.png |

Operation results:

|  |
| --- |
| This test will create a file called imagelist.yaml which contains the path to aloeGT.png. |

## Test case 62: imagelist\_reader

This code is an introductory example of using OpenCV to read a list of images from a YAML or XML file and process them. Its function is to read a file containing a list of image file paths, and then load and process these images one by one.

The main process is as follows:

1. **Command line parameter parsing** : The program obtains the path of the image list file through the command line parameters. If the path is not provided, the program will display a help message and exit.
2. **Read image list** : Use FileStorage to open the YAML or XML file, read the image path list and store it in vector<string>. If the file reading fails, the program will output an error message and exit.
3. **Process images** : Traverse the image path list, load each image in turn and process it. Here, the image is loaded in grayscale mode by default, and the path of the currently processed image is output to the console. If an image cannot be loaded, the program will output an error message and stop processing.
4. **Result output** : After each image is loaded and processed, the corresponding path will be printed out in the console to prompt the user of the image currently being processed.

This sample code provides a basic framework for OpenCV image processing work, showing how to read a list of images from a file and process them one by one.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Operation results:

|  |
| --- |
| This test will process the images listed in the imagelist.yaml file and output the processing information of each image in the console. |

## Test case sixty-three: imgcodecs\_jpeg

This code creates a large image made up of multiple small images and saves it as a JPEG file. Here is an overview of the main functions of the code:

1. **Input parameter check** : The code requires an output image file name to be passed in as a parameter. If the parameter is insufficient, the program will output usage instructions and exit.
2. **Initialize the image** : The code first creates a 160x160 blank image and draws a central circle in it. The radius of the circle gradually increases from 5, is drawn every 3 pixels, and a black border is drawn on the edge of the image.
3. **Create a large image to hold all the small images** : stitch all the processed images into a large framebuffer with a size of (160 \* 2, 160 \* 5), i.e., two rows and five columns of images.
4. **Copying of the original image** : In the first row of the framebuffer, copy the original image into each cell.
5. **JPEG compression and decompression** : In the second line of the framebuffer, the code compresses the original image using different JPEG sampling factors (411, 420, 422, 440, 444) in turn, and then decompresses the compressed image and stores it in the corresponding location.
6. **Save final image** : Save the entire framebuffer as an output file in JPEG format.
7. **Output saved information** : The program outputs the saved file name in the console.

Through this process, the program generates a comparison chart that contains the original image and its versions processed with different JPEG sampling factors. The final image shows the impact of different JPEG sampling factors on image quality.

The code related to displaying the image is commented out in the code, so the entire processing is completed in the background and the user cannot see the intermediate image processing process.

The output file will contain the original image and images with different sampling factors, making it easy for users to compare the differences between them.

Run the use case:

|  |
| --- |
| ./example\_cpp\_imgcodecs\_jpeg output\_image.png |

Operation results:

|  |
| --- |
| Output image saved as: output\_image.png |

## Test case 64: inpaint

This code demonstrates how to use OpenCV's inpainting function to repair damaged areas in an image. The main steps in the code are as follows:

Image loading: The program first loads a picture from a file. If the image cannot be opened, an error message is output and the program exits.

Create an inpaint mask: The program creates a single-channel mask (inpaintMask) of the same size as the original image. Initially, the mask is all 0 (black), indicating that there is no area that needs to be inpainted.

Simulating damaged areas: To demonstrate the repair function, the code fills white pixels in a manually specified area (a rectangle), which will be considered as damaged areas and need to be repaired. In the mask image, the corresponding areas are also filled with white.

Run the repair algorithm: Use the inpaint function to repair the image. This function uses the INPAINT\_TELEA algorithm to repair the damaged area based on the information of the surrounding area.

Save the repaired image: The repaired image is saved as inpainted\_image.jpg, and the console outputs the successful saving information.

Through this example, you can learn how to mark and repair damaged areas in an image to restore the integrity of the image.

Run the use case:

|  |
| --- |
| ./example\_cpp\_inpaint fruits.jpg |

Operation results:

|  |
| --- |
| Cool inpainting demo. Inpainting repairs damage to images by floodfilling the damage  with surrounding image areas.  Using OpenCV version 4.10.0-dev  Usage:  ./example\_cpp\_inpaint [image\_name -- Default fruits.jpg]  Inpainted image saved as: inpainted\_image.jpg |

## Test case sixty-five: intelligent\_scissors

This code demonstrates how to implement the "smart scissors" algorithm for contour tracking and segmentation in an image. The main process is as follows:

1. **Image loading and preprocessing** : The program first loads the input image and converts it into a grayscale image. Then it obtains the edge map through Canny edge detection and calculates the gradient direction and gradient magnitude of the image through the Sobel operator.
2. **Path calculation** : Based on image gradient and edge information, a priority queue and a custom pixel cost function are used to calculate the shortest path from the starting point to the target point. This path represents a contour in the image.
3. **Mouse interaction** : The program was originally designed as an interactive application where the user can draw contours on the image by clicking the mouse. However, this code comments out all parts related to the graphical interface and automatically starts the path calculation from the specified starting point.
4. **Result saving** : The final generated contour image is saved as output.png without manual intervention.

This code shows how to use image processing techniques to implement the smart scissors algorithm, which is suitable for automated image segmentation tasks.

Run the use case:

|  |
| --- |
| ./example\_cpp\_intelligent\_scissors fruits.jpg |

Operation results:

|  |
| --- |
| This test item will automatically process the image and save the processed image as output.png file. |

## Test case sixty-six: intersectExample

This code demonstrates how to use OpenCV's intersectConvexConvex function to calculate the intersection area of two convex polygons. The program creates various geometric shapes (such as rectangles and triangles), then calculates and draws their intersection area, and marks the intersection area on the image.

The main process is as follows:

1. **Shape creation** : Use the makeRectangle and makeTriangle functions to generate the vertex lists of rectangles and triangles that represent the boundaries of these shapes.
2. **Intersection calculation and drawing** : The drawIntersection function calculates the intersection of two convex polygons and draws the result on the image. If the input polygon is invalid (such as a non-convex polygon), the intersection result will be marked in red.
3. **Intersection scene display** : The program displays several different scenes, including rectangles with shared edges, triangles with vertices on the edges, and invalid input (non-convex polygons). The intersection result of each scene is plotted and the intersection area is marked.
4. **Result saving** : The final generated image is saved as intersections\_result.png, showing all calculated intersection shapes and corresponding areas.

This sample program uses different combinations of geometric shapes to intuitively demonstrate the effects and application scenarios of the intersectConvexConvex function in calculating the intersection of convex polygons.

Run the use case:

|  |
| --- |
| ./example\_cpp\_intersectExample |

Operation results:

|  |
| --- |
| This test will automatically process the image and save the processed image as intersections\_result.png file. |

## Test case sixty-seven: kalman

This code demonstrates how to use OpenCV's Kalman filter to track a point moving in a circle. The program simulates the movement of the point, predicts the position through the Kalman filter, measures the noise, and performs state correction to verify the effectiveness of the filter.

The main process is as follows:

1. **Initialization** : Create a 2D state Kalman filter, the state includes angle and angular velocity. Define the state transfer matrix, measurement matrix, process noise covariance and measurement noise covariance.
2. **Iterative tracking** : In each iteration, the program generates the location of the point based on the current state and uses the Kalman filter to make predictions. Noise is then added to simulate measurements, and the predictions are corrected using the measurements.
3. **Result drawing** : Draw the real points, measured points, predicted points and corrected points on the image and save the image as kalman\_result.png.
4. **Stop condition** : The program stops running after reaching the maximum number of iterations, demonstrating the effect of Kalman filtering in dynamic systems.

This example shows the power of Kalman filters in smoothing noisy measurements and dynamic tracking.

Run the use case:

|  |
| --- |
| ./example\_cpp\_kalman |

Operation results:

|  |
| --- |
| Example of c calls to OpenCV's Kalman filter.  Tracking of rotating point.  Point moves in a circle and is characterized by a 1D state.  state\_k+1 = state\_k + speed + process\_noise N(0, 1e-5)  The speed is constant.  Both state and measurements vectors are 1D (a point angle),  Measurement is the real state + gaussian noise N(0, 1e-1).  The real and the measured points are connected with red line segment,  the real and the estimated points are connected with yellow line segment,  the real and the corrected estimated points are connected with green line segment.  (if Kalman filter works correctly,  the yellow segment should be shorter than the red one and  the green segment should be shorter than the yellow one).  The program will run for a set number of iterations and then exit. This command will automatically process the image and save the processed image as kalman\_result.png file. The program will automatically exit after processing is completed. |

## Test case sixty-eight: kmeans

This code shows how to use OpenCV's kmeans function to cluster 2D points. The program generates a set of points randomly and applies the kmeans algorithm to assign these points to different clusters.

The main process is as follows:

1. **Randomly generated data** : The program generates randomly distributed point sets on a 500x500 image. The number of point sets and the number of clusters are randomly determined. The points in each cluster are generated according to a multi-Gaussian distribution.
2. **Apply K-means algorithm** : The program uses kmeans function to cluster the generated point set into up to 5 clusters and calculates the compactness of each cluster.
3. **Result drawing** : According to the clustering results, the program draws each point on the image and marks different clusters with different colors, and marks the cluster center on the image.
4. **Result saving** : The program runs 10 times, and each time the clustering results are saved as image files with file names from clusters\_1.png to clusters\_10.png.

This example shows how to cluster random data using the K-means algorithm and visualizes the clustering results.

Run the use case:

|  |
| --- |
| ./example\_cpp\_kmeans |

Operation results:

|  |
| --- |
| This command will automatically process the image and save the processed image files (e.g. clusters\_1.png, clusters\_2.png, etc.). The program will automatically exit after processing is complete.  Iteration: 1, Compactness: 801575  Iteration: 2, Compactness: 171295  Iteration: 3, Compactness: 359643  Iteration: 4, Compactness: 766363  Iteration: 5, Compactness: 437739  Iteration: 6, Compactness: 306076  Iteration: 7, Compactness: 428676  Iteration: 8, Compactness: 28007.5  Iteration: 9, Compactness: 376615  Iteration: 10, Compactness: 859590 |

## Test case sixty-nine: laplace

This code demonstrates how to use OpenCV's Laplacian function for edge detection. The program loads an image from a specified image file and applies different smoothing filters (Gaussian blur, mean blur, median blur) and the Laplacian operator in a loop to detect image edges.

The main process is as follows:

1. **Image loading** : The program loads the input image from the specified path and outputs the width and height information of the image.
2. **Smoothing Filtering and Laplacian Edge Detection** : In each iteration, the program applies the specified smoothing filter to the image and then uses the Laplacian operator to detect edges and generate the resulting image.
3. **Result saving** : The result image of each iteration is saved as laplacian\_<iteration>.png. The program runs a maximum of 100 iterations.

Through this example, users can observe the impact of different smoothing filters on the Laplacian edge detection effect.

Run the use case:

|  |
| --- |
| ./example\_cpp\_laplace -i=fruits.jpg |

Operation results:

|  |
| --- |
| Image fruits.jpg: width=512, height=480 |

## Test case seventy: letter\_recog

This code shows how to use the machine learning library (cv::ml) in OpenCV to train and test a variety of classifiers, including Random Trees, Boosting, Multilayer Perceptron (MLP), K Nearest Neighbor (KNN), Naive Bayes (NBayes), and Support Vector Machine (SVM). The code uses a letter recognition dataset for training and testing, which contains 20,000 samples, each representing a letter.

The main steps are as follows:

1. **Data reading** : Read feature data and labels (letters) from a file and store in a matrix.
2. **Classifier training** : Based on the classifier type selected by the user, the model is trained using the training data. If a pre-trained model path is specified, the model is loaded.
3. **Model testing and saving** : Calculate the recognition rate of the model on the training set and test set, and save the trained model to a file as needed.

The code also supports specifying the dataset path, model save path, and the type of classifier to be used through command line parameters.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Operation results:

|  |
| --- |
| ./example\_cpp\_letter\_recog |

## Test case 71: lkdemo

This code demonstrates the Lucas-Kanade method for optical flow tracking using OpenCV. The program reads frames from a video file and demonstrates the optical flow effect by detecting and tracking feature points. Key features include automatic initialization of tracking, manual addition or removal of feature points, and night mode.

**Code function overview:**

1. **Optical flow tracking** : Use goodFeaturesToTrack and calcOpticalFlowPyrLK functions to detect and track feature points.
2. **User interaction** : Supports adding feature points by clicking the mouse, and initializing tracking, clearing feature points, and switching night mode through shortcut keys.
3. **Video Processing** : The processed frames are saved as new video files, and support is provided for creating a video folder for saving the results.

**Main processing flow:**

* Initialize video capture and optical flow tracker.
* Feature points are detected and tracked in each frame and drawn on the frame.
* Save the processed frames to the specified video file.

Run the use case:

|  |
| --- |
| ./example\_cpp\_lkdemo ../../samples/data/vtest.avi |

Operation results:

|  |
| --- |
| This is a demo of Lukas-Kanade optical flow lkdemo(),  Using OpenCV version 4.10.0-dev  It uses a video file as input, which should be provided as a command-line argument.  Hot keys:  ESC - quit the program  r - auto-initialize tracking  c - delete all the points  n - switch the "night" mode on/off  To add/remove a feature point click it  Processed video will be saved at: lkdemo/output.avi |

## 测试用例七十二：logistic\_regression

This code demonstrates how to use Logistic Regression in OpenCV to train and test a simple binary classification model. The dataset contains images of handwritten digits 0 and 1, each flattened into a 28x28 matrix. The code divides the dataset into training and test sets and trains the logistic regression model. After training, the model predicts the test set and calculates the accuracy. Finally, the trained model is saved to an XML file and loaded from the file to verify its correctness.

Main process:

1. **Data loading** : Read image data and labels from XML files and divide them into training and test sets.
2. **Model training** : Use the logistic regression model for training, using batch gradient descent method.
3. **Prediction and evaluation** : Make predictions on the test set and calculate the prediction accuracy of the model.
4. **Model saving and loading** : Save the trained model to a file and reload the model from the file for prediction verification.

This code shows the complete process of basic machine learning model training, prediction, and persistence operations.

Run the use case:

|  |
| --- |
| ./example\_cpp\_logistic\_regression --data=../../samples/data/data01.xml |

Operation results:

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  /home/chenweijia/opencv/bin/cpp/../../samples/data/data01.xml contains digits 0 and 1 of 20 samples each, collected on an Android device  Each of the collected images are of size 28 x 28 re-arranged to 1 x 784 matrix  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  loading the dataset...read 40 rows of data  training/testing samples count: 20/20  training...done!  predicting...done!  original vs predicted:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]  accuracy: 95%  saving the classifier to NewLR\_Trained.xml  loading a new classifier from NewLR\_Trained.xml  predicting the dataset using the loaded classifier...done!  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]  accuracy: 95% |

## 测试用例七十三：lsd\_lines

This code demonstrates how to use OpenCV for line segment detection. The code is configured via command line parameters, allowing the user to select the input image, whether to use the Canny edge detector, whether to apply the standard refinement method of the line segment detector, whether to overlay the result on the original image, and specify the file name of the output image.

The main process is as follows:

1. **Parameter parsing** : Use CommandLineParser to parse the input parameters, including the image file path, refinement method selection, whether to use Canny detection, whether to overlay the results on the original image, and the output file path.
2. **Image loading** : Loads the input image and converts it to a grayscale image.
3. **Edge Detection (optional)** : Based on user selection, applies the Canny edge detector to highlight edges in the image.
4. **Line Segment Detection** : Detect line segments in the image using LineSegmentDetector, based on the refinement method selected by the user.
5. **Result display and saving** : Draw the detected line segments on the image and save the results to the specified file.

This code provides a flexible framework for detecting and saving the results of line segments from an image.

Run the use case:

|  |
| --- |
| ./example\_cpp\_lsd\_lines --input=../../samples/data/building.jpg --output=./result.jpg |

Operation results:

|  |
| --- |
| Usage: example\_cpp\_lsd\_lines [params]  -c, --canny (value:false)  use Canny edge detector  -h, --help (value:false)  show help message  -i, --input (value:../../samples/data/building.jpg)  input image  -o, --overlay (value:false)  show result on input image  -o, --output (value:./result.jpg)  output image  -r, --refine (value:false)  if true use LSD\_REFINE\_STD method, if false use LSD\_REFINE\_NONE method  It took 228.322 ms.  Result saved to ./result.jpg |

## Test case seventy-four: mask\_tmpl

This code demonstrates how to implement the "smart scissors" algorithm for contour tracking and segmentation in an image. The main process is as follows:

1. **Image loading and preprocessing** : The program first loads the input image and converts it into a grayscale image. Then it obtains the edge map through Canny edge detection and calculates the gradient direction and gradient magnitude of the image through the Sobel operator.
2. **Path calculation** : Based on image gradient and edge information, a priority queue and a custom pixel cost function are used to calculate the shortest path from the starting point to the target point. This path represents a contour in the image.
3. **Mouse interaction** : The program was originally designed as an interactive application where the user can draw contours on the image by clicking the mouse. However, this code comments out all parts related to the graphical interface and automatically starts the path calculation from the specified starting point.
4. **Result saving** : The final generated contour image is saved as output.png without manual intervention.

This code shows how to use image processing techniques to implement the smart scissors algorithm, which is suitable for automated image segmentation tasks.

Run the use case:

|  |
| --- |
| ./example\_cpp\_mask\_tmpl --input=../../samples/data/lena.jpg --t=../../samples/data/templ.png --m=../../samples/data/mask.png --o=./result.jpg |

Operation results:

|  |
| --- |
| Usage: example\_cpp\_mask\_tmpl [params]  --cm (value:3)  comparison method  -h, --help (value:false)  show help message  -i, --input (value:../../samples/data/lena.jpg)  input image  -m (value:../../samples/data/mask.png)  mask name  -o (value:./result.jpg)  output image name  -t (value:../../samples/data/templ.png)  template name  Result saved to ./result.jpg |

## Test case seventy-five: matchmethod\_orb\_akaze\_brisk

This code demonstrates how to implement the "smart scissors" algorithm for contour tracking and segmentation in an image. The main process is as follows:

1. **Image loading and preprocessing** : The program first loads the input image and converts it into a grayscale image. Then it obtains the edge map through Canny edge detection and calculates the gradient direction and gradient magnitude of the image through the Sobel operator.
2. **Path calculation** : Based on image gradient and edge information, a priority queue and a custom pixel cost function are used to calculate the shortest path from the starting point to the target point. This path represents a contour in the image.
3. **Mouse interaction** : The program was originally designed as an interactive application where the user can draw contours on the image by clicking the mouse. However, this code comments out all parts related to the graphical interface and automatically starts the path calculation from the specified starting point.
4. **Result saving** : The final generated contour image is saved as output.png without manual intervention.

This code shows how to use image processing techniques to implement the smart scissors algorithm, which is suitable for automated image segmentation tasks.

Run the use case:

|  |
| --- |
| ./example\_cpp\_matchmethod\_orb\_akaze\_brisk --image1=../../samples/data/basketball1.png --image2=../../samples/data/basketball2.png |

Running results (the log is too long, only part of it is shown):

|  |
| --- |
| Result saved to AKAZE-DESCRIPTOR\_KAZE\_UPRIGHT\_BruteForce.jpg  \*\*\*\*\*\*\*\*\*\*Match results\*\*\*\*\*\*\*\*\*\*\*\*  Index Index distance  in img1 in img2  225 212 0.00614143  425 400 0.00652844  228 216 0.00784745  432 407 0.00787756  285 270 0.00894252  259 244 0.0089759  413 388 0.00937356  1 0 0.00986802  286 272 0.00995928  105 91 0.0105124  115 102 0.0106676  486 464 0.0107331  287 273 0.0110073  233 219 0.0111014  3 1 0.0114316  108 94 0.0119987  291 277 0.0122909  288 274 0.01242  246 231 0.0124534  6 4 0.0131412  276 262 0.0153905  114 100 0.0155769  112 97 0.0160372  273 261 0.0160441  121 108 0.0160951  4 2 0.0163694  294 279 0.0166344  292 276 0.0176461  111 98 0.0179325  110 96 0.0185041  Result saved to AKAZE-DESCRIPTOR\_KAZE\_UPRIGHT\_BruteForce-L1.jpg e |

## Test case seventy-six: minarea

This code demonstrates how to use OpenCV to calculate the minimum bounding rectangle, triangle, and circle for a set of random points. The core flow of the code is as follows:

1. **Initialization and parameter parsing** : The program first parses the command line parameters, and the user can specify the file name of the output image.
2. **Random point generation** : The program generates a set of random points whose coordinates are limited to the central area of the image.
3. **Minimum enclosing figure calculation** :
   * Use the minAreaRect() function to calculate the minimum area bounding rectangle and get the four vertices of the rectangle.
   * Use minEnclosingTriangle() to calculate the minimum area enclosing triangle.
   * Use minEnclosingCircle() to calculate the center and radius of the minimum area enclosing circle.
4. **Draw and save results** : The program draws the generated points and their minimum enclosing rectangles, triangles and circles on the image. The final result is saved as the specified file.

This code demonstrates the basic application of geometric shape detection and processing in OpenCV.

Run the use case:

|  |
| --- |
| minarea ./example\_cpp\_minarea --output=./result.jpg |

Operation results:

|  |
| --- |
| This program demonstrates finding the minimum enclosing box, triangle or circle of a set  of points using functions: minAreaRect() minEnclosingTriangle() minEnclosingCircle().  Random points are generated and then enclosed.  Press ESC, 'q' or 'Q' to exit and any other key to regenerate the set of points.  Result saved to ./result.jpg |

## Test case seventy-seven: morphology2

This code demonstrates how to use morphological operations in OpenCV (including opening, closing, erosion and dilation) to process images. The core functions include:

1. **Command Line Argument Parsing** : The program accepts input image and output file names via command line arguments.
2. **Morphological operations** :
   * The OpenClose() function performs an opening or closing operation depending on the value of open\_close\_pos.
   * The ErodeDilate() function performs an erosion or dilation operation depending on the value of erode\_dilate\_pos.
   * Both operations use a structuring element (rectangle, ellipse, or cross) defined by element\_shape.
3. **Image processing** : The program processes the image in a loop, performing opening and closing operations and erosion and dilation operations respectively. The resulting image is saved to the specified file.
4. **Automatic exit** : To simplify the operation, the program exits the loop immediately after saving the processing results.

This code demonstrates the basic application of image morphology processing, which is one of the basic operations of image processing.

Run the use case:

|  |
| --- |
| ./example\_cpp\_morphology2 --image=../../samples/data/baboon.jpg --output=./result.jpg |

Operation results:

|  |
| --- |
| Open/Close result saved to open\_close\_./result.jpg  Erode/Dilate result saved to erode\_dilate\_./result.jpg |

## Test case seventy-eight: neural\_network

This code demonstrates how to use ANN\_MLP (Multilayer Perceptron Artificial Neural Network) in OpenCV for classification tasks. Here are the main steps and explanations of the code:

1. **Generate random training data** :
   * Use the randn() function to generate a 100x100 random matrix data, and the data follows a normal distribution.
   * Create a label response matrix where the first half of the samples are labeled as class 0 and the second half are labeled as class 1. This is a binary classification problem.
2. **Create the neural network** :
   * Define the hierarchy of the neural network. layerSizes includes the input layer (100 neurons), one hidden layer (20 neurons), and the output layer (2 neurons, corresponding to the two classes).
   * Use ANN\_MLP::create() to create a neural network and set the activation function to SIGMOID\_SYM and the training method to backpropagation (BACKPROP).
3. **Training the Neural Network** :
   * Use TrainData::create() to convert data and responses into training data format, and then call the network->train() method for training.
4. **predict** :
   * After training is completed, the neural network predicts a vector of all 1s and outputs the result.
   * The neural network then makes a prediction for each row in the training data and outputs the result. This can be used to check how well the network is trained.
5. **Output** :
   * The prediction results are output to the console in the form of a matrix.

This code example shows how to use OpenCV's machine learning module to create and train a simple multilayer perceptron network and use it to make predictions on data.

Run the use case:

|  |
| --- |
| ./example\_cpp\_neural\_network |

Running results (only part of the log is shown due to too many logs):

|  |
| --- |
| Predict one-vector:  [0.49815065, 0.49921253]  Predict training data:  [0.49815512, 0.4992235]  [0.49815148, 0.4992148]  [0.49816421, 0.49922886]  [0.49815488, 0.49921444]  [0.4981553, 0.49921337]  [0.49815708, 0.49922335]  [0.49815473, 0.49921861]  [0.49815747, 0.49921554]  [0.49815911, 0.49922082]  [0.4981544, 0.49921104]  [0.49815312, 0.49920845]  [0.49815834, 0.4992134]  .....  [0.49815196, 0.49921131]  [0.49814603, 0.49921173] |

## Test case seventy-nine: npr\_demo

This code demonstrates how to use OpenCV for template matching. Template matching is a technique for finding a template image in an image. This code supports the use of masks to enhance the accuracy of the match. The main steps are as follows:

1. **Parameter parsing** : Parse the input parameters through CommandLineParser to specify the input image, template image, mask image, matching method and output image file name. If the user requests help information, display the help and exit.
2. **Load image** : Load the input image, template and mask. If the loading fails, the corresponding error message will be output and the program will exit.
3. **Resize mask** : If the dimensions of the template and mask do not match, the program will resize the mask to the size of the template.
4. **Template matching** : Use the matchTemplate function to perform template matching, and the comparison method is specified by the user through command line parameters.
5. **Find the best matching position** : According to different matching methods (such as square difference, normalized square difference, etc.), find the best matching position and mark it with a green rectangle in the original image.
6. **Save result** : Save the marked image to the specified file.

This code provides a flexible framework for finding templates from an image and saving the matches to a file.

Run the use case:

|  |
| --- |
| ./example\_cpp\_npr\_demo --image=../../samples/data/lena.jpg --output=./result.jpg |

Running results (the red 1 and 2 need to be entered by the user, or select other options):

|  |
| --- |
| Edge Preserve Filter  ----------------------  Options:  1) Edge Preserve Smoothing  -> Using Normalized convolution Filter  -> Using Recursive Filter  2) Detail Enhancement  3) Pencil sketch/Color Pencil Drawing  4) Stylization  Press number 1-4 to choose from above techniques: 1  Press 1 for Normalized Convolution Filter and 2 for Recursive Filter: 2  Processed image saved to output.jpg |

## Test case 80: opencv\_version

This code is a simple command line program that outputs OpenCV version information and build configuration. The main functions of the code are as follows:

1. **Command Line Parameter Parsing** : Use cv::CommandLineParser to parse command line parameters. Supports two parameters: --help (or -h) and --build (or -b).
   * --help: Display help information.
   * --build: Output complete OpenCV build information.
2. **About information** : parser.about is used to set the program's descriptive information, which is displayed when the user requests help.
3. **Parameter handling** :
   * If the user requests help (--help), the program will output help information.
   * If an error occurs during command line parsing, the program will output an error message.
   * If the user uses the --build parameter, the program will call cv::getBuildInformation() to output detailed OpenCV build information.
   * If no arguments or unknown arguments are provided, the program will output the currently used OpenCV version.
4. **Output information** : Depending on the parsed parameters, the program will output build information, version information or help information respectively.

This code works fine for checking the installed OpenCV version as well as its build configuration.

Run the use case:

|  |
| --- |
| ./example\_cpp\_opencv\_version |

Operation results:

|  |
| --- |
| Welcome to OpenCV 4.10.0-dev |

## Test case 81: pca

This code demonstrates how to use OpenCV to perform principal component analysis (PCA) and observe the effect of PCA by changing the value of the preserved variance. The main functions of the code are as follows:

1. **Image reading and preprocessing** : Read images from the given image list and format them into matrices suitable for PCA input.
2. **PCA Execution** : Perform PCA on the image data using a specified variance preservation ratio (initial 95%). PCA compresses high-dimensional image data into a lower-dimensional feature space.
3. **Reconstruct and save images** : Map the PCA results back to the original space to generate reconstructed images that contain less information than the original images, and save these reconstructed images to disk.
4. **Automated testing** : The program automatically simulates different variance retention values (from 10% to 100%), observes the PCA effect, and saves the resulting image to the specified directory.

The whole process demonstrates the effect of PCA in data dimensionality reduction and reconstruction, and saves multiple reconstructed images with different retained variances for further analysis and comparison.

Run the use case:

|  |
| --- |
| ./example\_cpp\_pca images.txt output=./result |

Operation results:

|  |
| --- |
| Initial processed image saved to output=./result\_Reconstruction\_initial.jpg img1 - |

## Test case 82: peopledetect

This code demonstrates how to use OpenCV's HOG (Histogram of Oriented Gradients) descriptor for pedestrian detection. The main functions of the code are as follows:

1. **Initialize detector** : The program uses two predefined pedestrian detectors, one is the default HOG descriptor and the other is the Daimler pedestrian detector. These two detectors can be switched in the program.
2. **Video input and output settings** : The program can capture video from the camera in real time, or process pre-recorded video files and save the processed video as a new file. When saving, if the specified output directory does not exist, the program will try to create the directory.
3. **Pedestrian detection** : The program loops through the video frames and uses the selected HOG detector to detect pedestrians in the frame. The detected pedestrians will be marked with a green rectangle.
4. **Performance Timing and Display** : After each frame is processed, the program will calculate and display the frame rate (FPS) of the frame and the currently used detection mode.
5. **Video saving** : Each processed frame will be saved to the specified output video file.

With this code, users can detect pedestrians in videos in real time or offline and output new video files with pedestrians marked.

Run the use case:

|  |
| --- |
| ./example\_cpp\_peopledetect --video=../../samples/data/vtest.avi --output=./peopledetect/output.avi |

Operation results:

|  |
| --- |
| Processing video, output will be saved to: ./peopledetect/output.avi  Finished reading: empty frame  Processing complete. Video saved to: ./peopledetect/output.avi |

## Test case 83: phase\_corr

This code shows how to use OpenCV for video stabilization, specifically using the **phase correlation** method. The following are the main functions and workflow of the code:

1. **Command Line Parameter Parsing** : The code uses CommandLineParser to get the input video file path and output directory from the command line. If these two parameters are not provided, the program will report an error and terminate.
2. **Create output directory** : If the specified output directory does not exist, the program will use system commands to create the directory.
3. **Video reading** : The program opens the specified video file through the VideoCapture class. If the video file cannot be opened, the program reports an error and terminates.
4. **Video frame processing** :
   * Loop through the video frames and convert them to grayscale images.
   * When the first frame is initialized, a Hanning window is created for subsequent phase correlation calculations.
   * The phase correlation algorithm is used on the current frame and the previous frame to calculate the image translation.
   * If significant translation is detected (radius greater than 5), the program draws a circle and a line on the current frame indicating the direction and distance of the translation.
5. **Save processed frames** : The program saves the processed results of each frame as a PNG image. The file name is automatically generated according to the frame number and saved in the specified output directory.
6. **Output frame saving path** : Every time a frame is saved, the program will output the saving path in the console.

With this code, users can process a video frame by frame and visualize the translation changes between frames. This is very useful in application scenarios such as video stabilization and motion analysis.

Run the use case:

|  |
| --- |
| ./example\_cpp\_phase\_corr ../../samples/data/vtest.avi ./output.avi |

Running results (794 frames will be processed, the log is too long and only part of it is shown):

|  |
| --- |
| This sample demonstrates phase correlation for video stabilization.  Usage: example\_cpp\_phase\_corr [params] input output  input (value:../../samples/data/vtest.avi)  input video file  output (value:./output.avi)  output directory  Frame saved at: ./output.avi/frame\_0.png  Frame saved at: ./output.avi/frame\_1.png  Frame saved at: ./output.avi/frame\_2.png  Frame saved at: ./output.avi/frame\_3.png |

## Test case 84: points\_classifier

This code shows how to use various machine learning classifiers in OpenCV to classify an image and visualize the decision boundary. Here is an overview of the key parts and functionality of the code:

**Key features:**

1. **Input and output** : The output file path is obtained through command line parameters to save the image of the classification result. If the output file path is not provided, the program will prompt and exit.
2. **Data preparation** :
   * Demonstrate the training of a classifier using some predefined training points and categories (colors). These points simulate data points of different categories and are convenient for showing the decision boundary.
   * The prepare\_train\_samples() and prepare\_train\_data() functions are used to convert training points into training data for use by various classifiers.
3. **Multiple classifiers** :
   * It provides implementations of various machine learning models, including Naive Bayes (NBC), K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Decision Tree (DT), AdaBoost (BT), Random Forest (RF), etc.
   * Each classifier has a corresponding function (such as find\_decision\_boundary\_NBC) that is used to train the model and predict the category of the test sample. The prediction results are saved in the form of images.
4. **Decision boundary drawing** :
   * The predict\_and\_save() function iterates over each pixel of the image, predicts its category using the trained model, and displays it in the output image with the corresponding color.
5. **Main process** :
   * Initialize the image and classifier settings.
   * Demonstrates how to use different classifiers for training and prediction, and save the prediction results to the specified output file.

Run the use case:

|  |
| --- |
| ./example\_cpp\_points\_classifier ml/output.jpg |

Operation results:

|  |
| --- |
| Output saved to: ml/output.jpg  Output saved to: ml/output.jpg  Output saved to: ml/output.jpg  Output saved to: ml/output.jpg  Output saved to: ml/output.jpg  Output saved to: ml/output.jpg |

## Test case eighty-five: polar\_transforms

This code demonstrates how to use OpenCV to perform linear polar and log polar image transformations and save the results as image files. The following is a brief description of the key functions:

1. **Input and output** : The input video source (camera or video file) and the path to the output image file are specified through command line parameters. The program uses the VideoCapture object to read frames from the specified source.
2. **Polar coordinate transformation** :
   * The warpPolar() function is used to convert an image from a Cartesian coordinate system to a polar coordinate system, including linear polar coordinates and logarithmic polar coordinates.
   * The WARP\_INVERSE\_MAP flag is used to perform the inverse transformation, from a polar image back to a Cartesian image.
3. **Draw and save** :
   * Use drawMarker() to mark specific points on the original image and the polar coordinate image to visualize the effect of the transformation.
   * The generated images are saved as files, including the original image, linear polar coordinate image, log polar coordinate image and the results of their inverse transformation.
4. **Application scenarios** : This sample is suitable for image processing, video processing, and scenarios that require image coordinate system transformation.

The code can process real-time video streams and save converted images for subsequent analysis and visualization.

Run the use case:

|  |
| --- |
| ./example\_cpp\_polar\_transforms ../../samples/data/vtest.avi output |

Operation results:

|  |
| --- |
| This program illustrates usage of Linear-Polar and Log-Polar image transforms  Usage: example\_cpp\_polar\_transforms [params] input output   input (value:../../samples/data/vtest.avi) camera device number or video file path output (value:output) output image file path Images saved to: output\_\*.jpg Images saved to: output\_\*.jpg Images saved to: output\_\*.jpg Images saved to: output\_\*.jpg Images saved to: output\_\*.jpg Images saved to: output\_\*.jpg Images saved to: output\_\*.jpg Images saved to: output\_\*.jpg Images saved to: output\_\*.jpg |

## 测试用例八十六：qrcode

This code uses OpenCV to implement QR code detection and decoding functions, and supports detecting QR codes from image files or video streams (including real-time camera input). The code can specify input and output paths in the command line and configure the detection and decoding mode.

Key features include:

1. **QR code detection and decoding** : Supports standard contour detection methods and Aruco-based detection methods. You can choose to detect only the QR code boundary, or decode it at the same time.
2. **Video processing** : supports reading frames from cameras or video files for processing and displays FPS (frame rate) in real time. Users can switch between single QR code and multiple QR code detection modes in the video stream, and switch whether to decode.
3. **Image processing** : If the input is an image file, the program will detect the image multiple times and calculate the average FPS, and then save the processing results.
4. **Result saving** : The detected QR code information and image processing results can be saved to a specified directory. The user can also choose to save all processed frames or only the frames where the QR code was detected.

The program is suitable for scenarios where QR code detection is required in videos or images. It supports multiple detection modes and configurations, and the results can be saved for subsequent analysis.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Running results (the log is too long, only part of it is shown):

|  |
| --- |
| Press 'm' to switch between detectAndDecode and detectAndDecodeMulti  Press 'd' to switch between decoder and detector  Press ' ' (space) to save result into images  Press 'ESC' to exit  Run QR decoder on image: [768 x 576] (CV\_8UC3)  QR code is not detected  FPS: 1.87285  Saving QR code detection input: 'qrcode/qrcode\_output-00000\_input.png' ...  Saving QR code detection result: 'qrcode/qrcode\_output-00000.png' ...  Saved  ....  QR code is not detected  FPS: 3.94702  Saving QR code detection input: 'qrcode/qrcode\_output-00794\_input.png' ...  Saving QR code detection result: 'qrcode/qrcode\_output-00794.png' ...  Saved..........  End of video stream |

## 测试用例八十七：segment\_objects

This code demonstrates the basic method of background subtraction and connected component cleaning using OpenCV. The program captures frames through a video stream (or camera), applies background subtraction techniques to extract the foreground, and optimizes the foreground mask through morphological operations. It then identifies the largest connected region and draws its outline.

Key features include:

1. **Background Subtraction** : BackgroundSubtractorMOG2 is used to extract the foreground from the input video.
2. **Morphological processing** : Clean up the foreground mask through dilation and erosion operations to remove noise and small objects.
3. **Connected component analysis** : Identify and outline the largest connected component.
4. **Result saving** : Save each processed image frame to the segment\_objects directory, and the file name contains the frame number.

The program parses the input video path from the command line, and uses the default camera if it is not specified. Users can use the spacebar to pause or resume background learning (the code has commented out the parts related to keyboard interaction).

Run the use case:

|  |
| --- |
| ./example\_cpp\_segment\_objects ../../samples/data/vtest.avi |

Running results (the log is too long, only part of it is shown):

|  |
| --- |
| Saved: segment\_objects/segmented\_frame\_0.png  Saved: segment\_objects/segmented\_frame\_1.png  Saved: segment\_objects/segmented\_frame\_2.png  Saved: segment\_objects/segmented\_frame\_3.png  Saved: segment\_objects/segmented\_frame\_4.png  Saved: segment\_objects/segmented\_frame\_5.png  ..... Saved: segment\_objects/segmented\_frame\_794.png |

## Test case 88: select3dobj

This code demonstrates how to use OpenCV for camera calibration, image processing, and object 3D selection. Its main functions include:

1. **Camera calibration** : The program reads the camera's intrinsic matrix and distortion coefficient file to achieve image correction. The calibration result is used for subsequent image distortion correction.
2. **Chessboard corner detection** : The program uses a chessboard of known size for corner detection. The posture of the chessboard plane is calculated through these corner points, thus drawing a 3D box of the object on the image.
3. **3D box extraction** : Users can manually select the 3D space where the object is located through the program's operating interface. The program will draw a 3D box on the image and segment the object based on the user input.
4. **Image saving** : The program will save the processed images to the specified output directory for subsequent training or analysis.
5. **Video input** : The program supports input from video files or cameras, and realizes dynamic scene processing and object extraction through frame-by-frame processing.

With the above functions, the program can efficiently extract object images with 3D calibration information from videos or images and generate data sets for further processing.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Operation results:

|  |
| --- |
| The program outputs help information instead of executing the expected function:  This program's purpose is to collect data sets of an object and its segmentation mask.  It shows how to use a calibrated camera together with a calibration pattern to  compute the homography of the plane the calibration pattern is on. It also shows grabCut  segmentation etc.  ./example\_cpp\_select3dobj -w=<board\_width> -h=<board\_height> [-s=<square\_size>]  -i=<camera\_intrinsics\_filename> -o=<output\_prefix>  -w=<board\_width> Number of chessboard corners wide  -h=<board\_height> Number of chessboard corners height  [-s=<square\_size>] Optional measure of chessboard squares in meters  -i=<camera\_intrinsics\_filename> Camera matrix .yml file from calibration.cpp  -o=<output\_prefix> Prefix the output segmentation images with this  [video\_filename/cameraId] If present, read from that video file or that ID  Using a camera's intrinsics (from calibrating a camera -- see calibration.cpp) and an  image of the object sitting on a planar surface with a calibration pattern of  (board\_width x board\_height) on the surface, we draw a 3D box around the object. From  then on, we can move a camera and as long as it sees the chessboard calibration pattern,  it will store a mask of where the object is. We get successive images using <output\_prefix>  of the segmentation mask containing the object. This makes creating training sets easy.  It is best if the chessboard is odd x even in dimensions to avoid ambiguous poses.  The actions one can use while the program is running are:  ... |

Problem Analysis:

|  |
| --- |
| Command line parameter parsing failed: Outputting help information usually indicates that the program did not correctly parse the passed command line parameters. The files in the parameters all exist and can be accessed, but it still cannot run. |

## Test case eighty-nine: simd\_basic

This code shows how to use the SIMD (Single Instruction Multiple Data) instruction set in OpenCV to perform vectorized operations and check the SIMD features supported by the system. The main functions include:

1. **Macro definition check** : The program checks whether the compilation environment supports various SIMD extensions, such as CV\_SIMD128, CV\_SIMD256, etc., through the conditional compilation instruction #ifdef, and prints relevant information. These macro definitions determine whether the program can use SIMD instructions of a specific width.
2. **Memory size check** : If the system supports SIMD, the program will print the memory size of some common SIMD vector types (such as v\_uint8, v\_int32, and v\_float32). This helps to understand the space occupied by different data types in SIMD registers.
3. **Arithmetic Operation Demonstration** : The program demonstrates the basic addition operation of SIMD vectors, creates two vectors, and adds them together. The result is obtained and printed through the v\_get0 function, showing the execution effect of vectorized operations.
4. **Non-SIMD environment prompt** : If the SIMD instruction set is not available, the program will output a prompt message to remind the user to check the compilation target and build options.

This code is suitable for environments that need to check and use SIMD instruction sets to help developers optimize program performance.

Run the use case:

|  |
| --- |
| ./example\_cpp\_simd\_basic |

Operation results:

|  |
| --- |
| CV\_SIMD is defined: 0  CV\_SIMD\_WIDTH is defined: 16  CV\_SIMD128 is defined: 0  CV\_SIMD256 is defined: 0  CV\_SIMD512 is defined: 0  CV\_SIMD\_64F is defined: 0  CV\_SIMD\_FP16 is defined: 0  ================= sizeof checks =================  sizeof(v\_uint8) = 16  sizeof(v\_int32) = 16  sizeof(v\_float32) = 16  ================== arithm check =================  v\_get0(vx\_setall\_u8(10) + vx\_setall\_u8(45)) => 55  ===================== done ====================== |

## 测试用例九十：smiledetect

This code shows a program for smile detection using OpenCV. It uses Haar feature cascade classifiers to detect faces and smiles, and supports capturing input from a camera or video file. The main steps of the program are as follows:

1. **Initialization** : The program parses input parameters from the command line, such as the cascade classifier path for face and smile detection, scaling, whether to enable image flipping, etc.
2. **Load classifier** : Load the Haar feature cascade classifier for face and smile detection. If the loading fails, the program will output an error message and terminate.
3. **Capture video input** : The program can get input from a camera or a video file. If a camera index is specified, the video is captured from the corresponding camera; if a video file path is specified, the video is read from the file.
4. **Detect and Draw** : The program processes each video frame, first converting the image to grayscale and scaling it, then detecting the face position through a classifier. If the flip option is enabled, the flipped image is also detected. For each detected face region, the program also detects a smile within the region and draws a corresponding rectangle or circle depending on the detected smile strength.
5. **Output results** : The program outputs the intensity information of the smile detection through the command line and displays the detection results on the image.

This program is a basic computer vision application suitable for real-time smile detection and simple human-computer interaction applications.

Run the use case:

|  |
| --- |
| ./example\_cpp\_smiledetect --cascade=../../data/haarcascades/haarcascade\_frontalface\_alt.xml --smile-cascade=../../data/haarcascades/haarcascade\_smile.xml --scale=2.0 ../../samples/data/vtest.avi |

Operation results:

|  |
| --- |
| This program demonstrates the smile detector.  Usage:  ./example\_cpp\_smiledetect [--cascade=<cascade\_path> this is the frontal face classifier]  [--smile-cascade=[<smile\_cascade\_path>]]  [--scale=<image scale greater or equal to 1, try 2.0 for example. The larger the faster the processing>]  [--try-flip]  [video\_filename|camera\_index]  Example:  ./example\_cpp\_smiledetect --cascade="data/haarcascades/haarcascade\_frontalface\_alt.xml" --smile-cascade="data/haarcascades/haarcascade\_smile.xml" --scale=2.0  Using OpenCV version 4.10.0-dev  Video capturing has been started ...  NOTE: Smile intensity will only be valid after a first smile has been detected |

## Test case ninety-one: squares

This code shows how to use OpenCV to perform square detection in an image. The program uses pyramid scaling, Canny edge detection, contour detection, and contour approximation to identify the squares in the image.

The main steps are as follows:

1. **Image preprocessing** : Noise reduction by pyramid scaling (downscaling followed by upscaling) is performed, and then each color channel of the image is processed.
2. **Edge Detection** : Canny edge detection is applied to identify the edges in the image and the possible small holes between the edges are removed by dilation operation.
3. **Contour detection** : All contours are detected using the findContours function on the processed image and simplified using polygonal approximation (approxPolyDP).
4. **Square detection** : For each detected contour, the program checks whether it has 4 vertices, whether the area is large enough, and whether the contour is convex. Next, the cosine of each angle is calculated to ensure that all angles are close to 90 degrees. If the conditions are met, the contour is considered to be square and added to the results.
5. **Result saving** : The detected square will be drawn with green lines on the original image, and the result will be saved to the specified output directory.

When you run the program from the command line, you can pass in an image file as input, and the program will search for squares in the image and save the results.

Run the use case:

|  |
| --- |
| ./example\_cpp\_squares ../../samples/data/pic1.png |

Operation results:

|  |
| --- |
| Result saved to squares/result.png |

## Test case ninety-two: stereo\_calib

This code shows how to use OpenCV for stereo camera calibration, supporting both checkerboard and ChArUco checkerboard modes, and optionally using a predefined Aruco dictionary.

The main features of the program include:

1. **Read image list** : Read image file paths from XML or YML files for calibration.
2. **Stereo calibration** : Based on the given image list, detect the checkerboard or ChArUco checkerboard corner points in the image and calculate the camera's intrinsic and extrinsic parameters.
3. **Parameter saving** : Save the calibrated camera internal and external parameters to a file for later use.
4. **Image Correction** : Correct the image through stereo correction and save the result as an image file.
5. **Command line parameters** : Supports a variety of command line parameters. Users can specify the grid size, block size, Aruco dictionary type, etc. to flexibly adjust the calibration process.

The program can handle different calibration modes and can also display a corrected image of the calibration results. The code has error checking for certain situations to ensure the validity and consistency of the input.

Run the use case:

|  |
| --- |
| ./example\_cpp\_stereo\_calib -w=9 -h=6 -t=chessboard -s=1.0 -ms=0.5 -ad=DICT\_4X4\_50 -adf=None stereo\_calib.xml |

Operation results:

|  |
| --- |
| Reading file: ../../samples/data/left01.jpg  .Reading file: ../../samples/data/right01.jpg  .Reading file: ../../samples/data/left02.jpg  .Reading file: ../../samples/data/right02.jpg  .Reading file: ../../samples/data/left03.jpg  .Reading file: ../../samples/data/right03.jpg  .Reading file: ../../samples/data/left04.jpg  .Reading file: ../../samples/data/right04.jpg  .Reading file: ../../samples/data/left05.jpg  .Reading file: ../../samples/data/right05.jpg  .Reading file: ../../samples/data/left06.jpg  .Reading file: ../../samples/data/right06.jpg  .Reading file: ../../samples/data/left07.jpg  .Reading file: ../../samples/data/right07.jpg  .Reading file: ../../samples/data/left08.jpg  .Reading file: ../../samples/data/right08.jpg  .Reading file: ../../samples/data/left09.jpg  .Reading file: ../../samples/data/right09.jpg  .Reading file: ../../samples/data/left11.jpg  .Reading file: ../../samples/data/right11.jpg  .Reading file: ../../samples/data/left12.jpg  .Reading file: ../../samples/data/right12.jpg  .Reading file: ../../samples/data/left13.jpg  .Reading file: ../../samples/data/right13.jpg  .Reading file: ../../samples/data/left14.jpg  .Reading file: ../../samples/data/right14.jpg  .13 pairs have been successfully detected.  Running stereo calibration ...  done with RMS error=0.634207  average epipolar err = 0.445413  Intrinsic parameters saved to stereo\_calib/intrinsics.yml  Extrinsic parameters saved to stereo\_calib/extrinsics.yml  Rectified image saved to stereo\_calib/rectified\_0.png  Rectified image saved to stereo\_calib/rectified\_1.png  Rectified image saved to stereo\_calib/rectified\_2.png  Rectified image saved to stereo\_calib/rectified\_3.png  Rectified image saved to stereo\_calib/rectified\_4.png  Rectified image saved to stereo\_calib/rectified\_5.png  Rectified image saved to stereo\_calib/rectified\_6.png  Rectified image saved to stereo\_calib/rectified\_7.png  Rectified image saved to stereo\_calib/rectified\_8.png  Rectified image saved to stereo\_calib/rectified\_9.png  Rectified image saved to stereo\_calib/rectified\_10.png  Rectified image saved to stereo\_calib/rectified\_11.png  Rectified image saved to stereo\_calib/rectified\_12.png |

## Test case ninety-three: stereo\_match

This code shows how to use OpenCV for stereo matching to generate a disparity map and a 3D point cloud from two left and right images.

**Key features:**

1. **Stereo matching algorithm** : supports multiple stereo matching algorithms, including BM, SGBM, HH, HH4 and SGBM3way. Users can specify the algorithm through command line parameters.
2. **Image Processing** : Generates disparity maps from input left and right images, and optionally generates color disparity maps. Supports image scaling and display options.
3. **Camera calibration** : If the camera's intrinsic and extrinsic reference files are provided, the program will perform stereo calibration on the input image to improve matching accuracy.
4. **Point cloud generation** : Generate a 3D point cloud using the corrected disparity map and save it as an XYZ format file.
5. **Command line parameters** : Supports configuration of maximum disparity, block size, algorithm type, etc. through command line parameters to flexibly adjust the stereo matching effect.

**Error checking** : The program performs detailed error checking when processing command line parameters to ensure that the parameters entered by the user are valid.

Run the use case:

|  |
| --- |
| sudo ./example\_cpp\_stereo\_match ../../samples/data/left01.jpg ../../samples/data/right01.jpg --algorithm=sgbm --blocksize=5 --max-disparity=64 --scale=1.0 -i=calibration\_output.yml -e=calibration\_output.yml --no-display --color -o=disparity.png -p=pointcloud.txt |

Operation results:

|  |
| --- |
| /home/chenweijia/opencv/modules/core/src/convert\_c.cpp:113: error: (-215:Assertion failed) src.size == dst.size && src.channels() == dst.channels() in function 'cvConvertScale'| |

Problem Analysis:

|  |
| --- |
| Exception information: The error message indicates that when calling the cvConvertScale function, the size or number of channels of the input image src and the target image dst do not match. This function is usually used to scale or convert an image, so it is necessary to match the image size and number of channels.  Image size and channels: After checking, the size and number of channels of the input left.jpg and right.jpg images are the same. Therefore, the problem may not be directly caused by the size or number of channels of the input image. |

## Test case ninety-four: stitching

This code shows how to use OpenCV's Stitcher class to stitch images and save the resulting image.

**Key features:**

1. **Image stitching** : The code uses the Stitcher class to stitch the input images into panoramas or scans. The stitching mode can be specified by command line parameters. The default is panorama mode (PANORAMA), and scan mode (SCANS) is also supported.
2. **Processing command line parameters** : Supports multiple command line parameters, such as --d3 to divide the image into three parts to increase the stitching success rate, --mode to select the stitching mode, and --output to specify the name of the resulting image.
3. **Image reading and segmentation** : Pass multiple image paths through the command line, and the code will read these images. If the --d3 flag is enabled, each image will be split into three parts for stitching.
4. **Result saving** : After stitching is completed, the result image will be saved in the specified directory, and the default name is "result.jpg".

**Error Handling** : The program performs adequate error checking when processing command line arguments and reading images to ensure that the input images and arguments are valid.

Run the use case:

|  |
| --- |
| ./example\_cpp\_stitching --d3 --mode panorama --output result.jpg left01.jpg left02.jpg left03.jpg |

Operation results:

|  |
| --- |
| Stitching completed successfully. Result saved to stitching\_result/result.jpg |

## Test case ninety-five: stitching\_detailed

This code implements a detailed image stitching program, using several advanced functions of OpenCV, including feature matching, exposure compensation, image deformation, stitching, and blending. The following are the main functions and usage instructions of the code:

**Functional Overview**

1. **Feature extraction and matching** : The program supports the use of multiple feature detectors (such as ORB, AKAZE, SURF, SIFT) to extract image features and perform feature matching.
2. **Camera parameter estimation** : Estimate the camera parameters by calculating the transformation matrix between images.
3. **Image Warping** : Based on the estimated camera parameters, the image is geometrically warped to make it suitable for stitching.
4. **Exposure compensation** : Reduce the problem of inconsistent brightness during the stitching process by adjusting the exposure of the image.
5. **Stitching and Blending** : Stitch the deformed images and use multi-band blending technology to make the transition of the stitching area smoother.
6. **Time-series shooting support** : The program also supports outputting the stitching results of each frame as a time-series image, which is suitable for time-lapse photography.

**Instructions**

* Pass multiple image paths through the command line, and the program will automatically process these images and stitch them into a complete panorama.
* Supports a variety of configuration parameters, such as stitching mode, exposure compensation method, feature detector type, etc., which can be adjusted through command line parameters.
* The final stitching result will be saved in the stitching\_detailed directory, and the default file name is result.jpg.

Run the use case:

|  |
| --- |
| ./example\_cpp\_stitching\_detailed left01.jpg left02.jpg left03.jpg left04.jpg left05.jpg left06.jpg left07.jpg left08.jpg left09.jpg |

运行结果：

|  |
| --- |
| Finding features...  Features in image #1: 500  Features in image #2: 500  Features in image #3: 500  Features in image #4: 500  Features in image #5: 500  Features in image #6: 500  Features in image #7: 500  Features in image #8: 500  Features in image #9: 500  Finding features, time: 0.891909 sec  Pairwise matchingPairwise matching, time: 0.495647 sec  Need more images |

## 测试用例九十六：text\_skewness\_correction

This code shows how to correct tilted text in an image. The program receives a tilted image as input and outputs a corrected image. Here are the main steps of the code:

1. **Load image** : Read the input image from disk. If the loading fails, output an error message and terminate the program.
2. **Convert to grayscale image** : Use cvtColor to convert a color image to a grayscale image for subsequent processing.
3. **Binarization** : Binarize the grayscale image, set the foreground pixels to 255, and the background pixels to 0. This step uses the Otsu algorithm for threshold segmentation.
4. **Denoising** : Reduce random noise in the image through erosion operation, using rectangular structure elements for processing.
5. **Minimum enclosing rectangle** : Use the minAreaRect function to calculate the minimum enclosing rectangle of the text to obtain the rotation angle of the image.
6. **Adjust angle** : If the calculated angle is less than -45 degrees, add 90 degrees to correct the angle.
7. **Rotate Image** : Calculate the rotation matrix and apply it to the image to produce a rectified image.
8. **Save the result** : Create a directory called skewness\_correction and save the corrected image to this directory.
9. **Output angle information** : Print the correction angle on the console and mark the angle information on the image.

The program finally saves the corrected image as corrected\_image.png. All display-related functions are commented out so that it can be run in a non-graphical environment.

Run the use case:

|  |
| --- |
| ./example\_cpp\_asift --image1=../../samples/data/aero1.jpg --image2=../../samples/data/aero3.jpg |

Operation results:

|  |
| --- |
| [INFO] angle: 80.69 degrees  Image saved to skewness\_correction/corrected\_image.png |

## Test case ninety-seven: train\_HOG

This code shows how to use OpenCV to train and test an SVM classifier based on HOG (Histogram of Oriented Gradients) features. The following is the main flow of the code:

1. **Load images** : load images from the specified positive sample and negative sample directories. Positive samples are used for training, and negative samples are used to generate negative sample sets.
2. **Calculate HOG features** : Calculate HOG features for positive and negative samples, and use these features for SVM training. HOG features can effectively capture the gradient direction information of the image and are particularly suitable for object detection.
3. **Train SVM** : Use the calculated HOG features to train a linear SVM classifier. After training, the SVM model is saved to the specified file.
4. **Test SVM** : Test the trained SVM detector on a test dataset or video file and output the detection results.
5. **Model saving** : After training is completed, the SVM model is saved in a directory named train\_HOG. The model can be used for subsequent detection tasks.
6. **Double training** (optional): After the initial training, use the initially trained model to detect the false positive samples in the negative sample set, add these false positive samples to the negative sample set, and train the SVM again to improve the robustness of the detector.

The program combines image processing and machine learning methods for applications such as pedestrian detection and vehicle detection.

positive\_images/ in the same directory

Blender\_Suzanne1.jpg

Blender\_Suzanne2.jpg

negative\_images/

basketball1.png

basketball2.png

test\_images/

fruits.jpg

home.jpg

These image files are in samples/data

Run the use case:

|  |
| --- |
| #Training commands  ./example\_cpp\_train\_HOG -dw=64 -dh=128 -pd=positive\_images -nd=negative\_images -td=test\_images -fn=my\_detector.yml -d  #Test command  ./example\_cpp\_train\_HOG -t -fn=train\_HOG/my\_detector.yml -td=test\_images -tv=Megamind.avi |

Operation results:

|  |
| --- |
| #Training log  Positive images are being loaded......[done] 2 files.  Negative images are being loaded......[done] 2 files.  Negative images are being processed......[done] 2 files.  Histogram of Gradients are being calculated for positive images......[done] ( positive images count : 2 )  Histogram of Gradients are being calculated for negative images......[done] ( negative images count : 2 )  Training SVM......[done]  Testing trained detector on negative images. This might take a few minutes......[done]  Histogram of Gradients are being calculated for positive images......[done] ( positive count : 2 )  Histogram of Gradients are being calculated for negative images......[done] ( negative count : 3 )  Training SVM again......[done]  Trained SVM saved to train\_HOG/my\_detector.yml  Testing trained detector...  #测试日志  Testing trained detector... |

## Test case ninety-eight: train\_svmsgd

This code uses the SVMSGD (Support Vector Machine Stochastic Gradient Descent) algorithm in OpenCV to perform a simple linear classification demonstration. The program allows the user to add points to the image by clicking the mouse, and trains the SVM classifier in real time based on these points, and then plots the classification results.

Key features include:

1. **Data structure initialization** : The Data structure is used to store images, training samples, and corresponding labels. Create a black background image by specifying the image width and height.
2. **Train SVM** : The doTrain function uses the SVMSGD algorithm to train the added points and obtain the weights and offsets of the classifier.
3. **Drawing the classification line** : After adding new points and retraining the SVM, the findPointsForLine function calculates the decision boundary based on the trained weights and offsets and draws it on the image through the redraw function.
4. **Mouse interaction** : By clicking the mouse, the user can add positive samples (labeled as 1) with the left button and negative samples (labeled as -1) with the right button. After each click, the SVM will retrain and update the image.
5. **Result saving** : The program will save the final generated image to the train\_SVMSGD directory to ensure that the results can be persisted.

This program is suitable for demonstrating the working principle of the SVM classifier and its application to simple linear classification tasks.

Run the use case:

|  |
| --- |
| ./example\_cpp\_train\_svmsgd 841 594 |

Operation results:

|  |
| --- |
| Final image saved to train\_SVMSGD/final\_image.jpg |

## Test case ninety-nine: travelsalesman

This code implements a simulated annealing solver for a simple Traveling Salesman Problem (TSP). The program generates random city locations and optimizes the order in which these cities are visited to minimize the total travel distance using a simulated annealing algorithm.

**Key features:**

1. **TravelSalesman class** : Responsible for managing the location and path of the city. It includes functions for calculating the energy of the path (total distance), changing the path (perturbation operation of simulated annealing), and restoring to the previous state.
2. **Initialization of city positions** : Cities are randomly distributed in a circle in the image, and the initial path connects all cities in sequence.
3. **Draw the travel path** : The DrawTravelMap function is responsible for drawing the current city and path on the image, which is convenient for intuitively displaying the path optimization process.
4. **Simulated annealing algorithm** : By repeatedly perturbing the path and deciding whether to accept the new path based on the temperature and perturbation effect, the optimal solution is gradually found. The temperature is gradually reduced to reduce the amplitude of the perturbation.
5. **Result Saving** : The final optimized path image is saved to the specified directory to ensure that the result can be viewed persistently.

**Directions:**

The program receives two parameters: the number of cities and the file name of the output image. After running, the results will be saved in the travelsalesman directory.

Through this code, you can intuitively understand the application of simulated annealing algorithm in solving the traveling salesman problem.

Run the use case:

|  |
| --- |
| ./example\_cpp\_travelsalesman 40 final\_image.jpg |

Running results (the log is too long, only part of it is shown):

|  |
| --- |
| i=0 changesApplied=808216 temp=96.0596 result=5881.56 i=1 changesApplied=858262 temp=92.2745 result=4128.39 i=2 changesApplied=833786 temp=88.6385 result=6112.83 i=3 changesApplied=792061 temp=85.1458 result=6531.22 i=4 changesApplied=821249 temp=81.7907 result=4288.95 i=5 changesApplied=884042 temp=78.5678 result=3332.4 i=6 changesApplied=839431 temp=75.4719 result=3825.27 i=7 changesApplied=869839 temp=72.498 result=3317.85 i=8 changesApplied=845716 temp=69.6413 result=3297.11 i=9 changesApplied=904058 temp=66.8972 result=3155.89 i=10 changesApplied=881034 temp=64.2612 result=2944.53 i=11 changesApplied=873954 temp=61.729 result=2601.67 i=12 changesApplied=843252 temp=59.2966 result=2467.88 i=13 changesApplied=864529 temp=56.9601 result=2231.8 i=14 changesApplied=853781 temp=54.7157 result=2594.18 i=15 changesApplied=830185 temp=52.5596 result=3025.01 i=16 changesApplied=818816 temp=50.4886 result=2285.11 i=17 changesApplied=807015 temp=48.4991 result=2664.99 i=18 changesApplied=753353 temp=46.5881 result=3251.8 |

## Test case 100: tree\_engine

This code implements a simulated annealing solver for a simple Traveling Salesman Problem (TSP). The program generates random city locations and optimizes the order in which these cities are visited to minimize the total travel distance using a simulated annealing algorithm.

**Key features:**

1. **TravelSalesman class** : Responsible for managing the location and path of the city. It includes functions for calculating the energy of the path (total distance), changing the path (perturbation operation of simulated annealing), and restoring to the previous state.
2. **Initialization of city positions** : Cities are randomly distributed in a circle in the image, and the initial path connects all cities in sequence.
3. **Draw the travel path** : The DrawTravelMap function is responsible for drawing the current city and path on the image, which is convenient for intuitively displaying the path optimization process.
4. **Simulated annealing algorithm** : By repeatedly perturbing the path and deciding whether to accept the new path based on the temperature and perturbation effect, the optimal solution is gradually found. The temperature is gradually reduced to reduce the amplitude of the perturbation.
5. **Result Saving** : The final optimized path image is saved to the specified directory to ensure that the result can be viewed persistently.

**Directions:**

The program receives two parameters: the number of cities and the file name of the output image. After running, the results will be saved in the travelsalesman directory.

Through this code, you can intuitively understand the application of simulated annealing algorithm in solving the traveling salesman problem.

Run the use case:

|  |
| --- |
| ./example\_cpp\_tree\_engine -r=10 data.csv |

Operation results:

|  |
| --- |
| Reading in data.csv...  terminate called after throwing an instance of 'cv::Exception'  what(): OpenCV(4.10.0-dev) /home/chenweijia/opencv/modules/ml/src/data.cpp:602: error: (-215:Assertion failed) (!varTypesSet && vtypes[i] == rowtypes[i]) || (varTypesSet && (vtypes[i] == rowtypes[i] || rowtypes[i] == VAR\_ORDERED)) in function 'loadCSV' |

Problem Analysis:

|  |
| --- |
| The sample requires a data.csv file. I wrote a random generator myself, but it doesn't work. This error is caused by the data type specification (typespec) in OpenCV's TrainData::loadFromCSV function not matching the data type in the CSV file or not being set correctly. Specifically, the error indicates:  Data type mismatch: The variable types specified in the typespec parameter (vtypes) do not match the actual data types in the CSV file (rowtypes).  Variable types not set: When calling TrainData::loadFromCSV, the types of some variables were not specified correctly or were missing, causing the function to be unable to correctly identify the data type when parsing the CSV file.  The key parts of the error message are: Assertion failed: Indicates an assertion failure, meaning a condition in the code was not met. !varTypesSet && vtypes[i] == rowtypes[i]: In the case where the variable type is not set, vtypes[i] should match rowtypes[i].  varTypesSet && (vtypes[i] == rowtypes[i] || rowtypes[i] == VAR\_ORDERED): In the case where the variable types are set, vtypes[i] should match rowtypes[i], or rowtypes[i] is an ordered variable.  Error location: File: data.cpp Line number: 602  OpenCV version: 4.10.0-dev  This error usually occurs when the TrainData::loadFromCSV function tries to read and parse a CSV file, and the parsing fails and an exception is thrown because the data type specification does not match the actual data or is not set. |

## Test case 101: videocapture\_audio

This code uses OpenCV to read an audio file and extract and save the audio data. The program opens the audio file through VideoCapture and extracts the audio samples and saves them as a text file.

**Key features:**

1. **Audio file reading** : The program opens the audio file through the VideoCapture class and configures it to decode using FFmpeg.
2. **Audio data extraction** : extract audio data and store it in Mat matrix. The data of each audio channel is stored separately.
3. **Metadata output** : Before reading audio data, the program outputs basic information of the audio, such as sampling depth, samples per second, total number of channels, etc.
4. **Data saving** : Audio data is separated by channel and saved in text files. Each text file saves the sampling data of one audio channel.
5. **Directory management** : Before saving audio data, the program will check and create a storage directory to ensure that the data can be saved correctly.

**Directions:**

When you run the program, you need to provide an audio file as input. The program will extract the data from the audio file and save the data of each audio channel to a text file in the videocapture\_audio directory.

With this code, users can easily extract and save the data of audio files into text files, which is suitable for simple audio analysis and processing.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_audio ../../samples/data/ Megamind.avi |

Operation results:

|  |
| --- |
| CAP\_PROP\_AUDIO\_DATA\_DEPTH: 8U  CAP\_PROP\_AUDIO\_SAMPLES\_PER\_SECOND: 0  CAP\_PROP\_AUDIO\_TOTAL\_CHANNELS: 0  CAP\_PROP\_AUDIO\_TOTAL\_STREAMS: 0  Number of samples: 0 |

Problem Analysis:

|  |
| --- |
| The video file Megamind.avi can be opened successfully and the video frames can be read, but all the audio related properties such as CAP\_PROP\_AUDIO\_SAMPLES\_PER\_SECOND, CAP\_PROP\_AUDIO\_TOTAL\_CHANNELS and CAP\_PROP\_AUDIO\_TOTAL\_STREAMS are zero, resulting in no audio samples being extracted.  Audio parameters are zero: The log shows that the audio sample rate, number of audio channels, and total number of audio streams are all zero, which usually indicates that OpenCV cannot correctly identify or parse the audio stream in the AVI file.  Possible reasons: Unsupported audio format: The audio stream format in the Megamind.avi file may not be supported by the current OpenCV version.  Audio stream reading issue: Even if the audio stream exists, OpenCV may fail to read the data in the audio stream correctly, resulting in the audio parameters being zero.  Codec issues: If OpenCV is not correctly compiled or linked to a codec that handles a specific audio format, it may cause audio extraction to fail. |

## Test case 102: videocapture\_audio\_combination

This code uses OpenCV to read an audio file and extract and save the audio data. The program opens the audio file through VideoCapture and extracts the audio samples and saves them as a text file.

**Key features:**

1. **Audio file reading** : The program opens the audio file through the VideoCapture class and configures it to decode using FFmpeg.
2. **Audio data extraction** : extract audio data and store it in Mat matrix. The data of each audio channel is stored separately.
3. **Metadata output** : Before reading audio data, the program outputs basic information of the audio, such as sampling depth, samples per second, total number of channels, etc.
4. **Data saving** : Audio data is separated by channel and saved in text files. Each text file saves the sampling data of one audio channel.
5. **Directory management** : Before saving audio data, the program will check and create a storage directory to ensure that the data can be saved correctly.

**Directions:**

When you run the program, you need to provide an audio file as input. The program will extract the data from the audio file and save the data of each audio channel to a text file in the videocapture\_audio directory.

With this code, users can easily extract and save the data of audio files into text files, which is suitable for simple audio analysis and processing.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_audio\_combination ../../samples/data/ Megamind.avi |

Operation results:

|  |
| --- |
| Successfully opened file with backend: 0  CAP\_PROP\_AUDIO\_DATA\_DEPTH: 8U  CAP\_PROP\_AUDIO\_SAMPLES\_PER\_SECOND: 0  CAP\_PROP\_AUDIO\_TOTAL\_CHANNELS: 0  CAP\_PROP\_AUDIO\_TOTAL\_STREAMS: 0  Number of audio samples: 0  Number of video frames: 795 |

Operation results:

|  |
| --- |
| The video file Megamind.avi can be opened successfully and the video frames can be read, but all the audio related properties such as CAP\_PROP\_AUDIO\_SAMPLES\_PER\_SECOND, CAP\_PROP\_AUDIO\_TOTAL\_CHANNELS and CAP\_PROP\_AUDIO\_TOTAL\_STREAMS are zero, resulting in no audio samples being extracted.  Audio parameters are zero: The log shows that the audio sample rate, number of audio channels, and total number of audio streams are all zero, which usually indicates that OpenCV cannot correctly identify or parse the audio stream in the AVI file.  Possible reasons: Unsupported audio format: The audio stream format in the Megamind.avi file may not be supported by the current OpenCV version.  Audio stream reading issue: Even if the audio stream exists, OpenCV may fail to read the data in the audio stream correctly, resulting in the audio parameters being zero.  Codec issues: If OpenCV is not correctly compiled or linked to a codec that handles a specific audio format, it may cause audio extraction to fail. |

## Test case 103: videocapture\_basic

This code shows how to use OpenCV library for video capture and saving. It reads a video file and saves its contents as another video file.

**Key features:**

1. **Video file reading** : The program opens the specified video file through the VideoCapture class and checks whether the file is opened successfully.
2. **Video attribute acquisition** : The program obtains the video's frame width, height, frame rate (FPS) and other attributes to prepare for subsequent video writing.
3. **Video saving** : The program uses the VideoWriter class to save the read video frames into a new AVI file, and the saving path is videocapture\_basic/output.avi.
4. **Directory management** : Before saving the video, the program will check and create the storage directory videocapture\_basic to ensure that the output file can be saved correctly.
5. **Video Processing** : The program reads the video frame by frame and writes it to a new video file. The program stops processing if it encounters an empty frame or is terminated by the user.

**Directions:**

When you run the program, you need to provide a video file path as input. The program will read the video and save its contents as another video file.

Through this code, users can learn how to use OpenCV to perform basic video capture and save operations.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_basic ../../samples/data/vtest.avi |

Operation results:

|  |
| --- |
| Press any key to terminate  Video saved to videocapture\_basic/output.avidone |

## Test case 104: videocapture\_camera

This code implements basic video processing and saving using OpenCV. The program reads the specified video file and saves each frame into a new output video file after processing.

**Functional Overview:**

1. **Video file reading** : The program uses the VideoCapture class to open the specified video file and checks whether the file is opened successfully.
2. **Video attribute acquisition** : The program obtains video attributes such as frame width, height, and frame rate to prepare for subsequent video processing and saving.
3. **Video saving** : Use the VideoWriter class to save each processed frame into a new AVI video file. The save path is videocapture\_camera/output.avi.
4. **Frame processing** : The program reads the video frame by frame. If the processing option is enabled, the program will process the frames using the Canny edge detection algorithm; otherwise, the original frames are saved directly.
5. **Performance monitoring** : The program prints information such as the number of captured frames, average frame rate, and processing time per frame every 10 frames.
6. **Directory management** : Before saving the video, the program creates the storage directory videocapture\_camera to ensure that the output file can be saved correctly.

**Directions:**

When running the program, you need to provide a video file path as input. The program will read the video file and save each frame as a new video file after processing it.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_camera vtest.avi |

Operation results:

|  |
| --- |
| Opening video file: vtest.avi |

## Test case 105: videocapture\_gphoto2\_autofocus

This code implements an OpenCV-based video autofocus example, which is mainly used to process video files containing focus control. The code can read frames from video files, calculate image clarity using edge detection algorithms, automatically adjust the focus to obtain the best focus, and support saving the processed video.

**Functional Summary:**

1. **Input video reading and output video saving** : The program reads the specified video file and chooses whether to save the processed frames as a new video file based on user input.
2. **Autofocus logic** : The program uses edge detection to calculate the sharpness of the current frame and adjusts the focus according to the direction and step size of the focus change until the best focus is achieved. The user can adjust the focus manually or enable autofocus.
3. **Processing and saving** : During the processing, the program will process and save each frame. If the best focus is reached or the focus cannot be optimized further, the program will prompt the user to continue fine-tuning.
4. **Command line parameter parsing** : The program supports specifying input video, output file, frame rate, focal length, whether to display detailed information, and other settings through command line parameters.

**Instructions for use:**

* When running the program, you need to provide the input video file path.
* Optionally, the user can specify parameters such as output file path, frame rate, etc.
* The program automatically adjusts the focus, and the user can also manually control the focus through keyboard input.

**Directory Management:**

The program will create a directory videocapture\_gphoto2\_autofocus to store the generated output video files.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_gphoto2\_autofocus -o=output.avi -f=20 -d=0 -v ../../samples/data/Megamind.avi |

运行结果：

|  |
| --- |
| Found minimal focus step = 64  RATE=0.0354956  STATE RATE=0.0354956 STEP=1024 Last change=0 stepToLastMax=-1024  RATE=0.0349195  STATE RATE=0.0349195 STEP=-512 Last change=0 stepToLastMax=-512  RATE=0.0335201  STATE RATE=0.0335201 STEP=256 Last change=0 stepToLastMax=-768  RATE=0.0336201  STATE RATE=0.0336201 STEP=-128 Last change=0 stepToLastMax=-640  RATE=0.0331308  In focus, you can press 'f' to improve with small step, or 'r' to reset.  Video saved to videocapture\_gphoto2\_autofocus/output.avi |

## Test case 106: videocapture\_gstreamer\_pipeline

This code implements a simple OpenCV application for measuring the performance of video encoding and decoding. Users can choose different video backends (such as GStreamer, FFmpeg, etc.) and encoders (such as H.264, H.265, etc.) to process video files.

**Functional Overview:**

1. **Mode selection** : The user can select encoding or decoding mode via command line parameters. In decoding mode, the program will read and display frames from a video file. In encoding mode, the program will generate a composite video source and encode it to a file.
2. **Support for multiple video backends and encoders** : The program supports video processing through backends such as GStreamer and FFmpeg, and supports multiple video encoding formats such as H.264, H.265, MPEG, etc.
3. **Performance measurement** : The program uses the TickMeter class to measure the time to process video frames, thereby calculating the frame rate (FPS), and supports a fast measurement mode that automatically ends after processing a certain number of frames.
4. **Output management** : The program can save the processed video to the specified directory and print the saving path.

**Directions:**

* When running the program, the user needs to specify parameters such as mode, video backend, encoder, and video file path.
* In encoding mode, users can also specify video resolution and frame rate.

**Directory Management:**

The program will create a corresponding output directory according to the encoder type, such as videocapture\_gstreamer\_pipeline, and save the processed video files to this directory.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_gstreamer\_pipeline -f=vtest.avi -m=decode |

Operation results:

|  |
| --- |
| Mode: decode, Backend: default, File: /home/chenweijia/opencv/bin/cpp/../../samples/data/vtest.avi, Codec: h264  Attempting to create capture with backend: default and file: /home/chenweijia/opencv/bin/cpp/../../samples/data/vtest.avi  Created default capture ( /home/chenweijia/opencv/bin/cpp/../../samples/data/vtest.avi )  No more frames - break  795 frames in 10.5182 sec ~ 75.583 FPS (total time: 10.5456 sec)  Processed file saved at: /home/chenweijia/opencv/bin/cpp/../../samples/data/vtest.avi |

## Test case 107: videocapture\_image\_sequence

This code shows how to use OpenCV's VideoCapture interface to read a series of images and save them to a specified directory. The program allows the user to provide an image mask through command line parameters to read a sequence of images with a consistent naming format (for example, example\_%02d.jpg can read example\_00.jpg, example\_01.jpg, etc.).

**Functional Overview:**

1. **Command Line Parsing** : The program uses CommandLineParser to parse the command line arguments and accepts an image mask as input. If no mask is provided, the program will output an error message and exit.
2. **Image sequence reading** : Through the VideoCapture class, the program opens the image sequence and reads the image frame by frame. When reading fails (that is, reaching the end of the image sequence), the program stops reading.
3. **Image saving** : After each frame of the image is read, the program saves it to a newly created subdirectory videocapture\_image\_sequence, with the file name format being frame\_0.jpg, frame\_1.jpg, and so on.
4. **File directory management** : The program will check and create the output directory to ensure that the save path exists.
5. **Image display function comments** : To simplify the process, the image display part of the code is commented out. You can uncomment it and use it if necessary.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_image\_sequence ../../samples/data/left%02d.jpg |

Operation results:

|  |
| --- |
| This sample shows you how to read a sequence of images using the VideoCapture interface.  Usage: ./example\_cpp\_videocapture\_image\_sequence <image\_mask> (example mask: example\_%02d.jpg)  Image mask defines the name variation for the input images that have to be read as a sequence.  Using the mask example\_%02d.jpg will read in images labeled as 'example\_00.jpg', 'example\_01.jpg', etc.  Saved: videocapture\_image\_sequence/frame\_0.jpg  Saved: videocapture\_image\_sequence/frame\_1.jpg  Saved: videocapture\_image\_sequence/frame\_2.jpg  Saved: videocapture\_image\_sequence/frame\_3.jpg  Saved: videocapture\_image\_sequence/frame\_4.jpg  Saved: videocapture\_image\_sequence/frame\_5.jpg  Saved: videocapture\_image\_sequence/frame\_6.jpg  Saved: videocapture\_image\_sequence/frame\_7.jpg  Saved: videocapture\_image\_sequence/frame\_8.jpg  End of Sequence |

## Test case 108: videocapture\_microphone

This code shows how to use OpenCV's VideoCapture interface to capture frames from a video and save them as image files. The code receives a video file path through the command line parameters, reads the video, and displays some video attribute information. After reading the video frames, the program saves the last frame to the specified directory.

**Functional Overview:**

1. **Command Line Parameter Parsing** : Use CommandLineParser to parse the command line parameters and receive a video file path. If the video file path is not provided, the program will output an error message and exit.
2. **Video capture** : Open the specified video file through the VideoCapture class. The program will check whether the file is successfully opened and output the basic information of the video, including frame width, frame height and FPS.
3. **Video frame processing** : The program captures video frames and limits the processing time to 10 seconds. During this time, the program continuously grabs and retrieves video frames.
4. **Frame saving** : After the capture is completed, the program saves the last video frame to a newly created subdirectory videocapture\_microphone with the file name frame.jpg.
5. **Directory management** : The program uses the system function to create the output directory and ensure that the save path exists.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_microphone ../../samples/data/Megamind.avi |

Operation results:

|  |
| --- |
| This sample shows you how to capture audio data using the VideoCapture interface.  Usage: ./example\_cpp\_videocapture\_microphone <video\_file>  Video file should be an official OpenCV sample video file.  CAP\_PROP\_FRAME\_WIDTH: 720  CAP\_PROP\_FRAME\_HEIGHT: 528  CAP\_PROP\_FPS: 23.976  Saved: videocapture\_microphone/frame.jpg |

## Test case 109: videocapture\_obsensor

This code demonstrates how to use OpenCV's VideoCapture interface to read frames from a video file and generate a simulated depth image. The processed RGB image and depth image are saved to the specified directory.

**Functional Overview:**

1. **Command line parameter parsing** : The program receives a video file path as input. If a valid video file path is not provided, the program will output an error message and exit.
2. **Video Capture** : Open the video file through the VideoCapture class and check whether it is successfully opened. If it fails to open, the program will exit.
3. **Simulated depth map generation** : During the processing of each frame, the program generates a 16-bit unsigned integer matrix depthMap with the same size as the image, simulates the depth map data, and assigns it through the random function randu.
4. **Depth map processing** : Convert the simulated depth map to an 8-bit image for easier visualization and apply a pseudo-color map (e.g. using COLORMAP\_JET) to enhance the display.
5. **Save image** : The program saves the RGB image of each frame and the processed depth map to the directory videocapture\_obsensor. The directory is created using the system function.
6. **Processing limit** : The program processes the first 10 frames of video and saves the RGB image and depth map of each frame as JPEG and PNG format files respectively.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_obsensor ../../samples/data/Megamind.avi |

Operation results:

|  |
| --- |
| This sample shows you how to use OpenCV VideoCapture with a video file instead of an obsensor.  Usage: ./example\_cpp\_videocapture\_obsensor <video\_file>  Video file should be an official OpenCV sample video file.  Saved: videocapture\_obsensor/rgb\_0.jpg and videocapture\_obsensor/depth\_0.png  Saved: videocapture\_obsensor/rgb\_1.jpg and videocapture\_obsensor/depth\_1.png  Saved: videocapture\_obsensor/rgb\_2.jpg and videocapture\_obsensor/depth\_2.png  Saved: videocapture\_obsensor/rgb\_3.jpg and videocapture\_obsensor/depth\_3.png  Saved: videocapture\_obsensor/rgb\_4.jpg and videocapture\_obsensor/depth\_4.png  Saved: videocapture\_obsensor/rgb\_5.jpg and videocapture\_obsensor/depth\_5.png  Saved: videocapture\_obsensor/rgb\_6.jpg and videocapture\_obsensor/depth\_6.png  Saved: videocapture\_obsensor/rgb\_7.jpg and videocapture\_obsensor/depth\_7.png  Saved: videocapture\_obsensor/rgb\_8.jpg and videocapture\_obsensor/depth\_8.png  Saved: videocapture\_obsensor/rgb\_9.jpg and videocapture\_obsensor/depth\_9.png |

## Test case 110: videocapture\_openni

This code shows how to use OpenCV to read data from depth sensors (such as Kinect, XtionPRO, etc.) and save the results as image files. The program first parses the command line parameters, selects the image type to be processed, and opens the specified .oni video file or real-time device. Then, it reads and processes the data frame by frame and saves different image files according to the selected output type.

**Functional Overview:**

1. **Command line parsing** : Specify the image type to be read (such as depth map, RGB image, etc.) and other settings, such as whether to colorize the disparity map, through command line parameters.
2. **Device or video file reading** : Open the device or video file and verify whether it is opened successfully.
3. **Frame-by-frame processing** : In each frame, the program extracts the depth map, disparity map, effective depth mask, RGB image, grayscale image, and infrared image from the device based on the user selection.
4. **Image saving** : Each type of image will be saved in the videocapture\_openni directory and the saved file name will be output.
5. **Limit the number of frames** : The program exits after processing the first 10 frames to avoid an infinite loop.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_openni -r=../../samples/data/Megamind.avi |

Operation results:

|  |
| --- |
| Device opening ...  done.  Saved videocapture\_openni/disparity\_0.png  Saved videocapture\_openni/bgr\_0.jpg  Saved videocapture\_openni/disparity\_1.png  Saved videocapture\_openni/bgr\_1.jpg  Saved videocapture\_openni/disparity\_2.png  Saved videocapture\_openni/bgr\_2.jpg  Saved videocapture\_openni/disparity\_3.png  Saved videocapture\_openni/bgr\_3.jpg  Saved videocapture\_openni/disparity\_4.png  Saved videocapture\_openni/bgr\_4.jpg  Saved videocapture\_openni/disparity\_5.png  Saved videocapture\_openni/bgr\_5.jpg  Saved videocapture\_openni/disparity\_6.png  Saved videocapture\_openni/bgr\_6.jpg  Saved videocapture\_openni/disparity\_7.png  Saved videocapture\_openni/bgr\_7.jpg  Saved videocapture\_openni/disparity\_8.png  Saved videocapture\_openni/bgr\_8.jpg  Saved videocapture\_openni/disparity\_9.png  Saved videocapture\_openni/bgr\_9.jpg |

## Test case 111: videocapture\_realsense

This code shows how to use OpenCV to read data from a RealSense camera (or simulated video file) and save the processed results as image files. The program reads each frame from the specified video file, simulates the generation of depth map and infrared (IR) image, and finally saves these images to the specified directory.

**Functional Overview:**

1. **Command line parsing** : The program specifies the path of the video file to be read through the command line parameters. If the path is not provided, the program will output an error message and terminate.
2. **Video file reading** : Use the VideoCapture class to open the video file and verify whether it is opened successfully.
3. **Simulated depth map and IR image** : After each frame read, the program generates a simulated depth map (stored as a 16-bit unsigned integer) and a simulated infrared image (stored as an 8-bit unsigned integer).
4. **Image processing and saving** :
   * The depth map is processed by normalization and color mapping to convert it into a color image.
   * Save the processed RGB image, depth map, and infrared image to the videocapture\_realsense directory.
5. **Limit the number of frames** : The program automatically exits after processing the first 10 frames to avoid an infinite loop.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_realsense ../../samples/data/Megamind.avi |

Operation results:

|  |
| --- |
| This sample shows you how to use OpenCV VideoCapture with a RealSense camera. Usage: ./example\_cpp\_videocapture\_realsense <video\_file> Video file should be an official OpenCV sample video file. Saved: videocapture\_realsense/rgb\_0.jpg, videocapture\_realsense/depth\_0.png, videocapture\_realsense/ir\_0.png Saved: videocapture\_realsense/rgb\_1.jpg, videocapture\_realsense/depth\_1.png, videocapture\_realsense/ir\_1.png Saved: videocapture\_realsense/rgb\_2.jpg, videocapture\_realsense/depth\_2.png, videocapture\_realsense/ir\_2.png Saved: videocapture\_realsense/rgb\_3.jpg, videocapture\_realsense/depth\_3.png, videocapture\_realsense/ir\_3.png Saved: videocapture\_realsense/rgb\_4.jpg, videocapture\_realsense/depth\_4.png, videocapture\_realsense/ir\_4.png Saved: videocapture\_realsense/rgb\_5.jpg, videocapture\_realsense/depth\_5.png, videocapture\_realsense/ir\_5.png Saved: videocapture\_realsense/rgb\_6.jpg, videocapture\_realsense/depth\_6.png, videocapture\_realsense/ir\_6.png Saved: videocapture\_realsense/rgb\_7.jpg, videocapture\_realsense/depth\_7.png, videocapture\_realsense/ir\_7.png Saved: videocapture\_realsense/rgb\_8.jpg, videocapture\_realsense/depth\_8.png, videocapture\_realsense/ir\_8.png Saved: videocapture\_realsense/rgb\_9.jpg, videocapture\_realsense/depth\_9.png, videocapture\_realsense/ir\_9.png |

## Test case 112: videocapture\_starter

This program shows how to use OpenCV to capture frames from a video file, image sequence, or a camera connected to your computer and save them. Detailed help information is included in the code, explaining the purpose of the program and how to use it.

**Functional Overview:**

1. **Command line parsing** : The program accepts a command line parameter, which can be a video file path, image sequence format, or camera device number.
2. **Capture and Processing** :
   * The program attempts to open the specified video file, image sequence, or camera.
   * If the open fails, an error message is output and the program terminates.
   * Each captured frame will be saved to the specified directory, and a maximum of 10 frames will be processed.
3. **Frame saving** : Each frame image is saved in .jpg format, and the file name is automatically generated according to the frame number.
4. **Directory creation** : Use the system function to create a subdirectory to store the captured frames.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videocapture\_starter ../../samples/data/Megamind.avi |

Operation results:

|  |
| --- |
| Saved videocapture\_starter/frame000.jpg  Saved videocapture\_starter/frame001.jpg  Saved videocapture\_starter/frame002.jpg  Saved videocapture\_starter/frame003.jpg  Saved videocapture\_starter/frame004.jpg  Saved videocapture\_starter/frame005.jpg  Saved videocapture\_starter/frame006.jpg  Saved videocapture\_starter/frame007.jpg  Saved videocapture\_starter/frame008.jpg  Saved videocapture\_starter/frame009.jpg |

## Test case 113: videowriter\_basic

This program shows how to use OpenCV to read frames from a video file and save them as a new video file. The main functions of the code include opening a video file, capturing video frames, encoding frames and writing to an output video file.

**Code Overview:**

1. **Command line parameter parsing** :
   * The program accepts one command line argument, the path to the video file to be processed.
   * If no video file path is provided, the program will prompt the user for input and terminate execution.
2. **Video Capture and Inspection** :
   * Open the video file using VideoCapture and check if the file opens successfully.
   * Read the first frame from the video to determine the frame size and type, and check whether the read was successful.
3. **Video encoder and output files** :
   * Set the output video encoder to MJPG and the frame rate to 25 frames per second.
   * Create a subdirectory called videowriter\_basic to store the generated video files.
   * Use VideoWriter to initialize the video writer, ready to save the processed frames to the output.avi file.
4. **Frame capture and writing loop** :
   * Loop through the captured video frames and write each frame to the output video file. The number of frames processed is limited to 100.
5. **End of program** :
   * When processing is complete or an empty frame is captured, the program outputs the saved video file name and ends.

Run the use case:

|  |
| --- |
| ./example\_cpp\_videowriter\_basic ../../samples/data/Megamind.avi |

Operation results:

|  |
| --- |
| Writing videofile: ./videowriter\_basic/output.avi  Saved video file: ./videowriter\_basic/output.avi |

## Test case 114: warpPerspective\_demo

This code demonstrates how to use OpenCV to perform a perspective transform on an image and save the result as a new image file. Here is a brief description of the code:

**Functional Overview**

* **Perspective Transformation** : The code uses the perspective transformation function of OpenCV to transform the image to a new perspective based on the four corner points specified by the user.
* **Image processing** : Use the mouse to select and move corner points and observe the effect of the transformation in real time.
* **File saving** : Save the transformed image to the specified directory.

**Main Process**

1. **Initialization** : When the code starts, the help message is displayed and the input image is loaded. The default image file is right.jpg.
2. **Set initial corner points** : According to the size of the image, initialize four corner points to define the area of perspective transformation.
3. **Loop processing** : In the loop:
   * Draw the image and the border based on the locations of the corner points.
   * If the four corner points are determined, the perspective transformation matrix is calculated and applied to generate the new image.
   * Save the transformed image to the warpPerspective\_demo directory with the frame number in the file name.
4. **Exit condition** : Exit the loop after processing 10 frames.

Run the use case:

|  |
| --- |
| ./example\_cpp\_warpPerspective\_demo ../../samples/data/right01.jpg |

Operation results:

|  |
| --- |
| Hot keys:  ESC, q - quit the program  r - change order of points to rotate transformation  c - delete selected points  i - change order of points to inverse transformation  Use your mouse to select a point and move it to see transformation changes  Saved warped image: warpPerspective\_demo/warped\_image\_0.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_1.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_2.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_3.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_4.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_5.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_6.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_7.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_8.jpg  Saved warped image: warpPerspective\_demo/warped\_image\_9.jpg |

## Test case 115: watershed

This code shows how to use OpenCV's watershed() function to perform watershed segmentation on an image and save the result as an image file. The following is a brief description of the code:

**Functional Overview**

* **Image Load** : Loads an input image from a file.
* **Mouse interaction** : Use mouse operation to draw the marked area for segmentation on the image (mouse drawing has been simulated in this example).
* **Watershed algorithm** : Use the watershed algorithm to segment the marked area and generate a segmented image.
* **Result saving** : Save the segmented image to the specified directory.

**Main Process**

1. **Initialization** : Load the input image and convert it to grayscale to create the marker mask.
2. **Mouse events** : Set up mouse callback functions to allow the user to draw markers on the image. Drawing operations are simulated in the example.
3. **Segmentation processing** :
   * Detect the outlines of the markers and generate a segmentation marker matrix.
   * Segmentation is performed using the watershed algorithm and each region is assigned a different color.
4. **Result output** : The generated segmented image is superimposed on the original image and saved in the watershed directory.

Run the use case:

|  |
| --- |
| ./example\_cpp\_watershed ../../samples/data/fruits.jpg |

Operation results:

|  |
| --- |
| This program demonstrates the famous watershed segmentation algorithm in OpenCV: watershed()  Usage:  ./example\_cpp\_watershed [image\_name -- default is fruits.jpg]  Hot keys:  ESC - quit the program  r - restore the original image  w or SPACE - run watershed segmentation algorithm  (before running it, \*roughly\* mark the areas to segment on the image)  (before that, roughly outline several markers on the image)  execution time = 26.5309ms  Saved watershed image: watershed/watershed\_transform.png |