VStabiliserOpenCv C++ library

v4.0.3

Table of contents

Overview

C++ library **VStabiliserOpenCv** (further – library) is designed for 2D and 3D digital video stabilization (horizontal offset, vertical offset and rotation). The library is used in camera systems for video vibration compensation. The library is written in C++ (C++17 standard) and uses the OpenCV library (version 4.5 and higher) to perform various operations (**cv::dft**, **cd::idft**, **cv::warpAffine** etc.). The library has a simple programming interface and a minimal number of parameters. The library inherits interface from **VStabiliser** class which provides flexible, customizable parameters and can be easily integrated into systems of any complexity. Additionally the demo application depends on open source **SimpleFileDialog** library (provide file dialog functions to open video files). The library supports varies pixel formats (**RBG24**, **BGR24**, **GRAY**, **YUV24**, **YUYV**, **UYVY**, **NV12**, **NV21**, **YV12**, **YU12**, listed in **Frame** class) and performs calculations in a single computational thread. The library is supplied as source code only.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	22.09.2020	First version.
1.1.0	22.09.2020	- Library interface updated. - Performance improved.
1.2.0	06.11.2020	- Added parameters for limiting the possible offset and rotation of the video frame. - Added the ability to scale video frames to increase the speed.
1.3.0	15.12.2020	- The speed of image rotation operation is increased by 40%. - Bugs fixed.
1.4.0	19.01.2021	- Added “transparent borders” mode.
1.5.0	08.07.2021	- The naming of the methods changed. - Code refactored. - Added method to get string of current library version.

Version	Release date	What's new
2.0.0	04.01.2022	- Calculation speed is increased by 30%. - Added support of new pixels formats: YUV, YUY2(YUYV) (2D stabilization only), UYVY (2D stabilization only) and NV12 (2D stabilization only). - Reduced number of adjustable parameters for ease of use. - Redesigned programming interface. - The mechanism of changing library parameters was redesigned. - Added the ability to use third-party libraries to rotate the image. - Fixed the bug with checking the offset and rotation limitations. - Fixed the bug of picture jumping when it is impossible to calculate the parameters of displacements on a particular frame of video.
3.0.0	10.05.2022	- Simplified software interface: access to software library properties via text-value key replaced by simple enumeration. - The error of transformation matrix values return has been fixed. - Redesigned control protocol parser: simplified interface and eliminated additional third-party dependencies.
3.1.0	12.06.2023	- Added constant offsets (X, Y and Rotation).
4.0.0	04.08.2023	- Changed programming interface according to new **VStabiliser** interface class. - Added new calculation algorithm based on FFT. Calculation speed increased **80%** in comparison with previous method. - Added auto calculation of filter parameters. - Added support for all RAW pixel format listed in **Frame** class.
4.0.1	10.08.2023	- Added support for python.
4.0.2	13.11.2023	- Frame class updated.
4.0.3	27.12.2023	- VStabiliser interface class updated. - getParams(...) method usage fixed in test applications. - Demo application updated (removed dependency from VSourceOpenCv library).

Library files

The library is supplied only by source code. The user is given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ------------------------ main CMake file
3rdparty ------------------------------ folder with 3rdparty libraries
    CMakeLists.txt -------------------- CMake file for 3rdpary folder
    VStabiliser ----------------------- files of VStabiliser interface library
src ----------------------------------- folder with library source code
    CMakeLists.txt -------------------- CMake file
    VStabiliserOpenCv.h --------------- main library header file
    VStabiliserOpenCvVersion.h -------- header file with library version
    VStabiliserOpenCvVersion.h.in ----- service CMake file to generate version file
    VStabiliserOpenCv.cpp ------------- C++ implementation file
demo ---------------------------------- folder for demo application files
    CMakeLists.txt -------------------- CMake file for demo app
    3rdaprty -------------------------- folder with 3rdparty libraries
        CMakeLists.txt ---------------- CMake file for 3rdparty folder
        SimpleFileDialog -------------- file dialog service library
    main.cpp -------------------------- source C++ file of demo app
benchmark ----------------------------- folder for benchmark app
    CMakeLists.txt -------------------- CMake file for benchmark app
    main.cpp -------------------------- source C++ file of bechmark app
cwrapper ------------------------------ folder for C-wrapper library
    CMakeLists.txt -------------------- CMake file for wrapper library
    CExternWrapper.cpp ---------------- source C++ file for creating shared lib
examples ------------------------------ folder with examples
    CMakeLists.txt -------------------- CMake file to include examples
    BgrExample ------------------------ example for BGR24 pixel format
    BgrWithoutCopyingExample ---------- example for BGR24 pixel format
    GrayExample ----------------------- example for GRAY pixel format
    Nv12Example ----------------------- example for NV12 pixel format
    Nv21Example ----------------------- example for NV21 pixel format
    PythonWrapperExample--------------- example for python support
    RgbExample ------------------------ example for RGB24 pixel format
    UyvyExample ----------------------- example for UYVY pixel format
    Yu12Example ----------------------- example for YU12 pixel format
    YuvExample ------------------------ example for YUV24 pixel format
    YuyvExample ----------------------- example for YUYV pixel format
    Yv12Example ----------------------- example for YV12 pixel format
```

VStabiliserOpenCv library depends on open source **VStabiliser** (provides interface for video stabiliser) which depends on open source **Frame** library (provides video frame structure and pixel formats description) and open source **ConfigReader** library (provides methods to work with JSON file and structures). Additionally library demo application depends on open source **SimpleFileDialog** (provides dialog to open files).

Key features and capabilities

Table 2 - Key features and capabilities.

Parameter and feature	Description
Programming language	C++ (standard C++17) using the OpenCV library (version 4.5 and higher).
Supported OS	Compatible with any operating system that supports the C++ compiler (C++17 standard) and the OpenCV library (version 4.5 and higher).
Number of stabilization axes	The library stabilizes video on three axes (3D stabilization): vertical, horizontal and rotation. Depends on algorithm type set by user.

Parameter and feature	Description
Supported pixel formats	RBG24, BGR24, GRAY, YUV24, YUYV (only 2D stabilisation), UYVY (only 2D stabilisation), NV12 (only 2D stabilisation), NV21 (only 2D stabilisation), YV12 (only 2D stabilisation), YU12 (only 2D stabilisation).
Allocated memory	The minimum amount of dynamically allocated memory is equal to the data size of 3 video frames in Y800 format (one byte per pixel in grayscale).
Maximum and minimum video frame size	Up to 30% of the video frame size vertically and horizontally. The allowable offsets are set by the user in the library parameters.
Maximum compensated frame rotation	Up to 60 degrees in any direction. The allowable rotation is set by the user in the library parameters.
Calculation speed	The processing time per video frame depends on the computing platform used and depends on algorithm type set by user. The processing time per video frame can be estimated with the demo application and benchmark application (provided by request). It is possible to scale video frames to provide higher calculation speed.
Implemented algorithms	Three types of stabilisation algorithm are implemented in the library: - **2D based on FFT**. Fastest algorithm but only for 2D stabilisation. Works stable for low light conditions and for low contrast images. - **2D based on optical flow**. Gives good accuracy but lower speed as 2D FFT and requires contrast objects on video. - **3D based on optical flow**. Gives best accuracy but lower speed as 2D FFT and requires contrast objects on video.
Boresight correction	The library provides method to add constant offsets for result image: horizontal offset (pixels), vertical offset (pixels) and rotation angle (radians).

Note: The values given in the table are applied to the concept of video frame(s) and pixel(s).

Benchmark results

In order to evaluate the calculation speed of the **VStabiliserOpenCv** library on a particular processor and operating system, the **VStabiliserOpenCvBenchmark** test program is provided. It allows you to test the library with different parameters. Additionally, a test video file is provided to ensure uniform testing conditions across different platforms. The chosen test video contains high vibration and a large number of contrast objects, creating the same conditions for evaluating the two different algorithms (based on FFT and based on optical flow) implemented in the library. This test video provides the most challenging conditions for the stabilisation algorithms. For scenarios with lower vibrations, the processing time can be lower as well.

Table 2 includes the processing time for one video frame on different hardware platforms (with default installation of the OpenCV library via the command: **sudo apt-get install libopencv-dev**) and with different parameters (video resolution, algorithm type, and scale factor for processing). If you have a special OpenCV installation with hardware-optimized functions (**cv::dft**, **cd::idft** and **cv::warpAffine**), the processing time can be significantly reduced. Here's an example of a video frame from the test video file:

Table 2 - Processing time (**msec**) for one frame of test video for different hardware platforms (**2D FFT** - 2D algorithm based on FFT, **2D OF** - 2D algorithm based on optical flow, **3D OF** - 3D algorithm based on optical flow) and different parameters (video resolution, algorithm type and scale factor for processing).

intel i7-13700H									
frame size / type / scale	**2D FFT / 1**	**2D FFT / 2**	**2D FFT / 4**	**2D OF / 1**	**2D OF / 2**	**2D OF / 4**	**3D OF / 1**	**3D OF / 2**	**3D OF / 4**
1920x1080	3.5	3.3	1.2	11.0	5.3	2.0	12.0	8.0	3.5
1280x720	3.3	1.0	1.0	5.0	2.5	1.3	5.5	2.5	1.4
640x512	2.3	0.5	0.5	2.5	1.2	1.0	3.0	1.6	1.2
Raspberry PI 4B									
frame size / type / scale	**2D FFT / 1**	**2D FFT / 2**	**2D FFT / 4**	**2D OF / 1**	**2D OF / 2**	**2D OF / 4**	**3D OF / 1**	**3D OF / 2**	**3D OF / 4**
1920x1080	29.0	28.0	9.0	37.0	17.0	11.0	49.0	32.0	28.0
1280x720	27.0	6.6	6	20.0	10.0	6.5	29	20	16
640x512	27.0	6.4	5.8	10.0	6.0	4.3	15.0	11.0	9.0
Jetson Orin NX									
frame size / type / scale	**2D FFT / 1**	**2D FFT / 2**	**2D FFT / 4**	**2D OF / 1**	**2D OF / 2**	**2D OF / 4**	**3D OF / 1**	**3D OF / 2**	**3D OF / 4**
1920x1080	13.2	13	3.9	16.5	7	3.3	24	16.2	13.6

intel i7-13700H									
1280x720	12.0	3.0	3.0	8.6	4.0	2.0	12.7	8.2	6.2
640x512	11.8	2.9	2.9	4.1	2.1	1.0	6.1	4.1	3.1

How to choose right algorithm

VStabiliserOpenCv library includes two types of algorithms to calculate offsets between the previous and current video frames: an algorithm based on FFT (Fourier transform) and an algorithm based on optical flow (2D or 3D). Each algorithm has pros and cons. If the video has a lot of contrasting objects, it's better to choose the algorithm based on optical flow. Algorithms based on optical flow generally take more time for processing than the algorithm based on FFT. When you need 3D stabilization, you have only one option - the 3D algorithm based on optical flow. When you need only 2D stabilization (horizontal and vertical), you can choose the algorithm based on optical flow (if you have contrasting objects in the video) or the algorithm based on FFT (robust to low contrast environments and much faster). For low contrast environments, algorithms based on optical flow may not work reliably. To choose the right algorithm, use the demo application to check how it works on your video. Here's an example of two images: on the left - a typical situation when algorithms based on optical flow don't work properly (choose 2D FFT algorithm), on the right - a high contrast video where all algorithms will work well.

Python support

The library is fully supported in the Python programming language. Developers can now incorporate our library's features into their Python codebase with ease, promoting a more streamlined development process. Simple guidance on incorporating our library into your Python code:

1. Build the shared library **CWrapperVStabiliserOpenCv** to create .dll file. This library serves as a bridge, wrapping the **VStabiliserOpenCv** class from C++ to C language. Minor modifications to function parameters were required, as detailed in **cwrapper/CExternWrapper.cpp**. CWrapperVStabiliserOpenCv.dll file can be used in any environment that supports .dll files.
2. Import CWrapperVStabiliserOpenCv.dll file directly in Python with the use of **ctypes** built-in python library:

```python
import ctypes
c_library = ctypes.CDLL(CWrapperVStabiliserOpenCv.dll)
```

3. After loading CWrapperVStabiliserOpenCv library, all of its functions become accessible in Python. Before using these functions, it's necessary to define proper return and argument types for each function that will be used. Here's how:

```
import numpy as np
int_type = ctypes.c_int
stabilizer_type = ctypes.POINTER(ctypes.c_char)
image_array_type = np.ctypeslib.ndpointer(dtype=np.uint8, ndim=channels_number,
                                          flags='C_CONTIGUOUS')

c_library.createVStabiliserOpenCv.restype = stabilizer_type
c_library.deleteVStabiliserOpenCv.argtypes = [stabilizer_type]
c_library.stabilise.argtypes = [stabilizer_type, int_type, int_type,
                                int_type, int_type, image_array_type,
                                image_array_type]
```

4. With the preliminary setup complete, you can now utilize the primary feature of the VStabiliserOpenCv library: the **stabilise()** function. The provided example employs the Python OpenCV library for video input handling and display. To begin, create an instance of the VStabiliserOpenCv class and remember to destroy it after performing the calculations.

```
# Create instance of VStabiliserOpenCv class, it will be passed to all functions.
VStabiliserOpenCv = c_library.createVStabiliserOpenCv()
# Create OpenCV video capture object to load images for stabilisation.
import cv2
video_capture = cv2.VideoCapture('video/test.mp4')
# Prepare input image informations.
width = int(video_capture.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(video_capture.get(cv2.CAP_PROP_FRAME_HEIGHT))
# Assign format according to cr::video::Fourcc enum class.
bgr_fourcc = 1
data_size = width * height * channels_number
# Initialize input and ouput images.
input_frame_ndarray = np.zeros((height, width, channels_number),
                               dtype=np.uint8)
output_frame_ndarray = np.zeros((height, width, channels_number),
                                dtype=np.uint8)
while True:
    success, input_frame_ndarray = video_capture.read()
    if not success:
            break
    c_library.stabilise(VStabiliserOpenCv, width, height, data_size,
                        bgr_fourcc, input_frame_ndarray,
                        output_frame_ndarray)

# Remember to destroy VStabiliserOpenCv object after calculations.
c_library.deleteVStabiliserOpenCv(VStabiliserOpenCv)
```

5. A comprehensive example can be found in the source code under **examples/PythonWrapperExample/python_wrapper_example.py**. Necessary methods available in the **VStabiliserOpenCv** class are accessible for use in your Python project. When passing arguments to C functions, ensure to use the appropriate integer values corresponding to the C++ enum classes. You

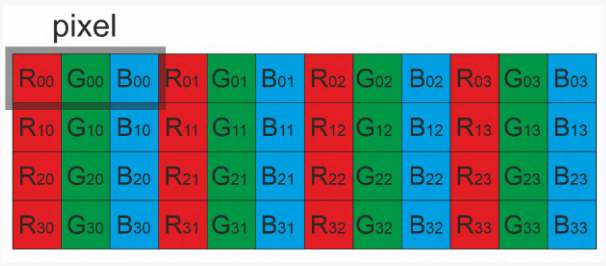
can find a look-up table in "python_wrapper_example.py".

Supported pixel formats

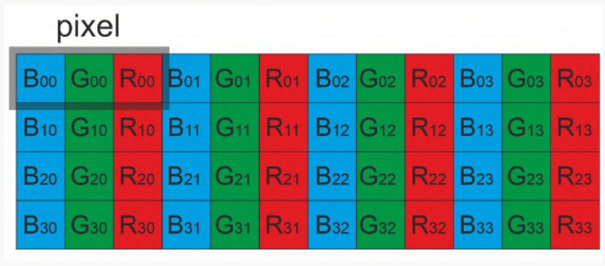
[Frame](#) library which included in **VStabiliserOpenCv** library contains **Fourcc** enum which defines supported pixel formats (**Frame.h** file). **VStabiliserOpenCv** library supports RAW pixel formats. **Fourcc** enum declaration:

```cpp
enum class Fourcc
{
    /// RGB 24bit pixel format.
    RGB24 = MAKE_FOURCC_CODE('R', 'G', 'B', '3'),
    /// BGR 24bit pixel format.
    BGR24 = MAKE_FOURCC_CODE('B', 'G', 'R', '3'),
    /// YUYV 16bits per pixel format.
    YUYV  = MAKE_FOURCC_CODE('Y', 'U', 'Y', 'V'),
    /// UYVY 16bits per pixel format.
    UYVY  = MAKE_FOURCC_CODE('U', 'Y', 'V', 'Y'),
    /// Grayscale 8bit.
    GRAY  = MAKE_FOURCC_CODE('G', 'R', 'A', 'Y'),
    /// YUV 24bit per pixel format.
    YUV24  = MAKE_FOURCC_CODE('Y', 'U', 'V', '3'),
    /// NV12 pixel format.
    NV12  = MAKE_FOURCC_CODE('N', 'V', '1', '2'),
    /// NV21 pixel format.
    NV21  = MAKE_FOURCC_CODE('N', 'V', '2', '1'),
    /// YU12 (YUV420) - Planar pixel format.
    YU12 = MAKE_FOURCC_CODE('Y', 'U', '1', '2'),
    /// YV12 (YVU420) - Planar pixel format.
    YV12 = MAKE_FOURCC_CODE('Y', 'V', '1', '2'),
    /// JPEG compressed format.
    JPEG  = MAKE_FOURCC_CODE('J', 'P', 'E', 'G'),
    /// H264 compressed format.
    H264  = MAKE_FOURCC_CODE('H', '2', '6', '4'),
    /// HEVC compressed format.
    HEVC  = MAKE_FOURCC_CODE('H', 'E', 'V', 'C')
};
```

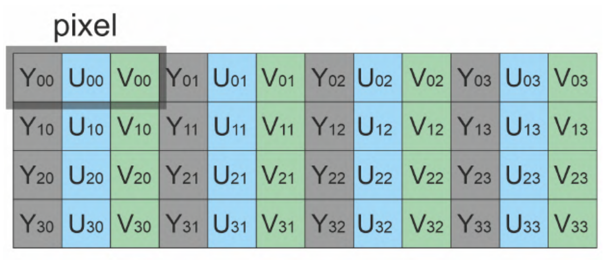
Table 3 - Bytes layout of supported RAW pixel formats. Example of 4x4 pixels image.



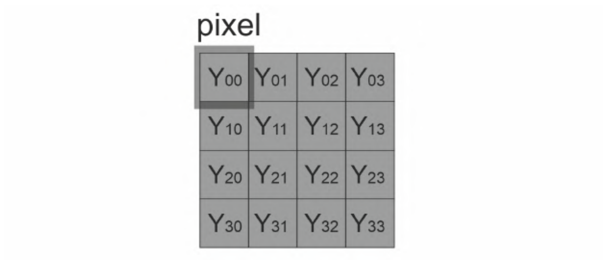
RGB24



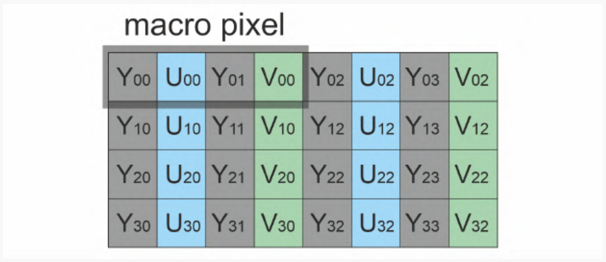
BGR24



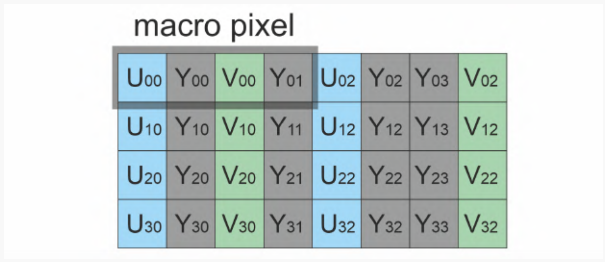
YUV24



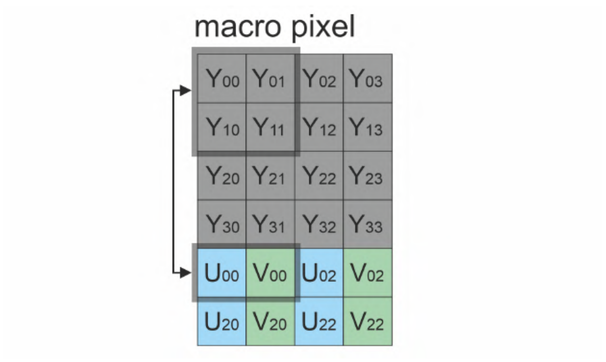
GRAY



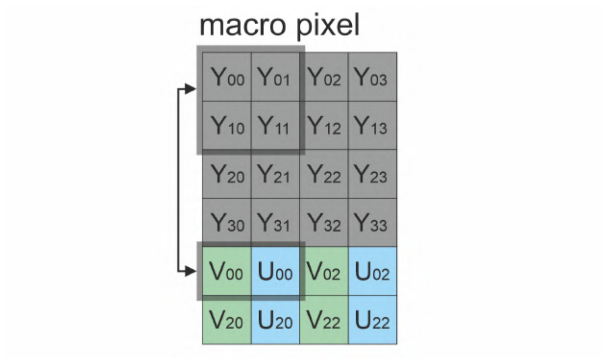
YUYV



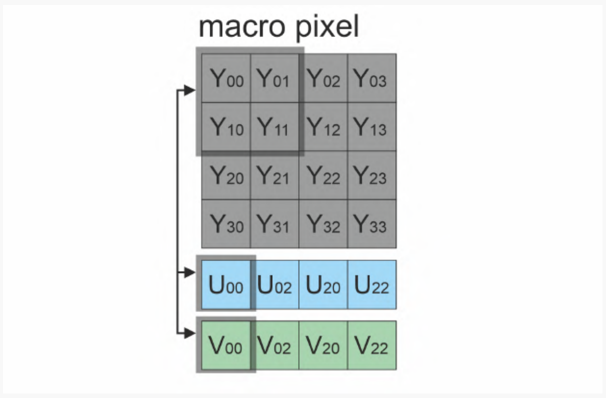
UYVY



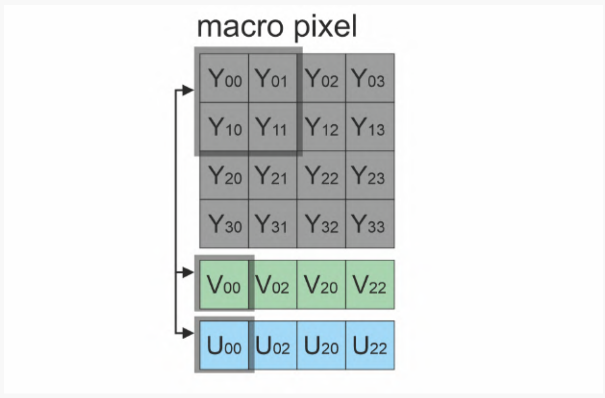
NV12



NV21



YU12



YV12

Library principles

The stabilization algorithm compensates horizontal displacement, vertical displacement and the rotation of video frames, taking into account the constant motion of the camera. The algorithm consists of the following sequential steps:

1. Obtaining the source video frame and converting it to Y800 format (grayscale).
2. Calculation offsets between current video frame and previous video frame (based on FFT or optical flow depends on parameters).
3. To compensate for the constant camera movement, the components of the camera constant motion (estimated by algorithm) are subtracted from the calculated transformation parameters.
4. The resulting transformation matrix is applied to the current image to compensate displacements and rotations.

The library delivered as source code only. To use the library, the developer must include library files in his project. It is also necessary to connect the OpenCV library version 4.5 or higher to the project. The sequence of using the library is as follows:

1. Include library files in the project.
2. Connect the OpenCV library to the project.
3. Create an instance of the VStabiliserOpenCv C++ class.
4. If necessary, change the default library parameters by calling the setParam(...).
5. Create Frame class objects for input and output video frames.
6. Call the stabilise(...) method to stabilise video frame.

VStabiliserOpenCv class description

VStabiliserOpenCv class declaration

VStabiliserOpenCv.h file contains **VStabiliserOpenCv** class declaration. VStabiliserOpenCv class inherits interface from **VStabiliser** interface class. Class declaration:

```
class VStabiliserOpenCv: public VStabiliser
{
public:
    /// Class constructor.
    VStabiliserOpenCv();

     /// Class destructor.
    ~VStabiliserOpenCv();

    /// Get string of current VStabiliserOpenCv class version.
    static std::string getVersion();

    /// Init all video stabiliser parameters by params structure.
    bool initVStabiliser(cr::vstab::VStabiliserParams& params) override;

    /// Set value to parameter with given id.
    bool setParam(cr::vstab::VStabiliserParam id, float value) override;

    /// Get parameter with given id.
    float getParam(cr::vstab::VStabiliserParam id) override;

    /// Get params.
    void getParams(VStabiliserParams& params) override;

    /// Execute command.
    bool executeCommand(cr::vstab::VStabiliserCommand id) override;

    /// Stabilise video frame.
    bool stabilise(cr::video::Frame& src, cr::video::Frame& dst) override;

    /// Get offsets: horizontal, vertical and rotation.
    void getOffsets(float& dX, float& dY, float& dA) override;

    /// Decode and execute command.
    bool decodeAndExecuteCommand(uint8_t* data, int size) override;
};
```

getVersion method

getVersion() method return string of current version of **VStabiliserOpenCv** class. Method declaration:

```
static std::string getVersion();
```

Method can be used without **VStabiliserOpenCv** class instance. Example:

```
cout << "VStabiliserOpenCv version: " << VStabiliserOpenCv::getVersion() << endl;
```

Console output:

```
VStabiliserOpenCv version: 4.0.3
```

initVStabiliser method

initVStabiliser(...) method intended to initialize video stabiliser parameters by set of parameters. Method copy all video stabiliser parameter to internal variables. Method declaration:

```
bool initVStabiliser(cr::vstab::VStabiliserParams& params) override;
```

Parameter	Description
params	Parameters class (see **VStabiliserParams** class description).

Returns: TRUE if parameters were accepted or FALSE if not.

setParam method

setParam(...) method intended to change video stabiliser parameter. Method declaration:

```
bool setParam(cr::vstab::VStabiliserParam id, float value) override;
```

Parameter	Description
id	Parameter ID according to **VStabiliserParam** enum.
value	Parameter value. Depends on parameter ID.

Returns: TRUE if param was accepted or FALSE if not.

getParam method

getParam(...) method intended to get video stabiliser parameter value. Method declaration:

```
float getParam(cr::vstab::VStabiliserParam id) override;
```

Parameter	Description
id	Parameter ID according to **VStabiliserParam** enum.

Returns: Parameter value or -1.0f if this param not supported.

getParams method

getParams(...) method designed to obtain all video stabiliser parameters. The library provides thread-safe **getParams(...)** method call. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

```
void getParams(VStabiliserParams& params) override;
```

Parameter	Description
params	Video stabiliser parameters class object (VStabiliserParams).

executeCommand method

executeCommand(...) method intended to execute video stabiliser command. Method declaration:

```
bool executeCommand(cr::vstab::VStabiliserCommand id) override;
```

Parameter	Description
id	Command ID according to **cr::vstab::VStabiliserCommand** enum.

Returns: TRUE if the command was executed or FALSE if not.

stabilise method

stabilise(...) method performs video stabilisation. Method declaration:

```
bool stabilise(cr::video::Frame& src, cr::video::Frame& dst) override;
```

Parameter	Description
src	Source frame. The methods accepts only RAW frame data (not compressed pixel formats, see description of **Frame** class). VStabiliserOpenCv supports following pixel formats: RGB24 - 2D and 3D stabilisation. BGR24 - 2D and 3D stabilisation. GRAY - 2D and 3D stabilisation. YUV24 - 2D and 3D stabilisation. YUYV - only 2D stabilisation will be performed even if set 3D algorithm type. UYVY - only 2D stabilisation will be performed even if set 3D algorithm type. NV12 - only 2D stabilisation with 2 pixels accuracy will be performed even if set 3D algorithm type. NV21 - only 2D stabilisation with 2 pixels accuracy will be performed even if set 3D algorithm type. YV12 - only 2D stabilisation with 2 pixels accuracy will be performed even if set 3D algorithm type. YU12 - only 2D stabilisation with 2 pixels accuracy will be performed even if set 3D algorithm type.
dst	Result frame. The pixel format of the result frame will be the same as source frame. If stabilisation disabled (param **MODE** set to 0) the library will copy data from source frame to result frame. If output frame not initialised the library will initialise it.

Returns: TRUE if video frame processed or FALSE in case any errors.

getOffsets methods

getOffsets(...) method returns horizontal offset (pixels), vertical offset (pixels) and rotation angle (radians) applied to last processed video frame (**stabilise** method). Method declaration:

```
void getOffsets(float& dX, float& dY, float& dA) override;
```

Parameter	Description
dX	Reference to output value of horizontal offset (pixels) implemented to last processed video frame.
dY	Reference to output value of vertical offset (pixels) implemented to last processed video frame.
dA	Reference to output value of rotational angle (radians) implemented to last processed video frame.

encodeSetParamCommand method of VStabiliser interface class

encodeSetParamCommand(...) static method designed to encode command to change any parameters of remote video stabiliser. To control a stabiliser remotely, the developer has to develop his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **VStabiliser** class contains static methods for encoding the control commands. The **VStabiliser** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeSetParamCommand(...)** designed to encode SET_PARAM command. Method declaration:

```
static void encodeSetParamCommand(uint8_t* data, int& size, VStabiliserParam id, float
value);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.
id	Parameter ID according to **VStabiliserParam enum**.
value	Parameter value.

SET_PARAM command format (11 bytes):

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	0x02	Major version of VStabiliser class.

Byte	Value	Description
2	0x04	Minor version of VStabiliser class.
3	id	Parameter ID **int32_t** in Little-endian format.
4	id	Parameter ID **int32_t** in Little-endian format.
5	id	Parameter ID **int32_t** in Little-endian format.
6	id	Parameter ID **int32_t** in Little-endian format.
7	value	Parameter value **float** in Little-endian format.
8	value	Parameter value **float** in Little-endian format.
9	value	Parameter value **float** in Little-endian format.
10	value	Parameter value **float** in Little-endian format.

encodeSetParamCommand(...) is static and used without **VStabiliser** class instance. This method used on client side (control system). Example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Random parameter value.
float outValue = (float)(rand() % 20);
// Encode command.
VStabiliser::encodeSetParamCommand(data, size, VStabiliserParam::INSTANT_X_OFFSET,
outValue);
```

encodeCommand method of VStabiliser interface class

encodeCommand(...) static method designed to encode video stabiliser action command. To control a stabiliser remotely, the developer has to develop his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **VStabiliser** class contains static methods for encoding the control commands. The **VStabiliser** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeCommand(...)** designed to encode COMMAND command (action command). Method declaration:

```
static void encodeCommand(uint8_t* data, int& size, VStabiliserCommand id);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 7 bytes.
id	Command ID according to VStabiliserCommand enum.

COMMAND format (7 bytes):

Byte	Value	Description
0	0x00	SET_PARAM command header value.
1	0x02	Major version of VStabiliser class.
2	0x04	Minor version of VStabiliser class.
3	id	Command ID **int32_t** in Little-endian format.
4	id	Command ID **int32_t** in Little-endian format.
5	id	Command ID **int32_t** in Little-endian format.
6	id	Command ID **int32_t** in Little-endian format.

encodeCommand(...) is static and used without **VStabiliser** class instance. This method used on client side (control system). Encoding example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Encode command.
VStabiliser::encodeCommand(data, size, VStabiliserCommand::ON);
```

decodeCommand method of VStabiliser interface class

decodeCommand(...) static method of VStabiliser interface class designed to decode command on video stabiliser side. To control a stabiliser remotely, the developer has to develop his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the VStabiliser interface class contains static method to decode input command (commands should be encoded by methods **encodeSetParamsCommand(...)** or **encodeCommand(...)**). The VStabiliser class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). Method declaration:

```
static int decodeCommand(uint8_t* data, int size, VStabiliserParam& paramId,
VStabiliserCommand& commandId, float& value);
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Should be 11 bytes (for SET_PARAMS) or 7 bytes for (COMMAND).
paramId	Video stabiliser parameter ID according to **VStabiliserParam** enum. After decoding SET_PARAM command the method will return parameter ID.
commandId	VStabiliser command ID according to **VStabiliserCommand** enum. After decoding COMMAND the method will return command ID.

Parameter	Description
value	Video stabiliser parameter value after decoding SET_PARAM command.

Returns: 0 - in case decoding COMMAND, **1** - in case decoding SET_PARAM command or **-1** in case errors.

Data structures

VStabiliserParam enum

VStabiliserParam enum lists video stabiliser parameters to set or to obtain. **VStabiliserParam** enum declared in **VStabiliser.h** file of the VStabiliser library. Enum declaration.

```
enum class VStabiliserParam
{
    /// Scale factor. Value depends on implementation. Default:
    /// If 1 the library will process original frame size, if 2
    /// the library will scale original frame size by 2, if 3 - by 3.
    SCALE_FACTOR = 1,
    /// Maximum horizontal image shift in pixels per video frame. If image shift
    /// bigger than this limit the library should compensate only X_OFFSET_LIMIT
    /// shift.
    X_OFFSET_LIMIT,
    /// Maximum vertical image shift in pixels per video frame. If image shift
    /// bigger than this limit the library should compensate only Y_OFFSET_LIMIT
    /// shift.
    Y_OFFSET_LIMIT,
    /// Maximum rotational image angle in radians per video frame. If image
    /// absolute rotational angle bigger than this limit the library should
    /// compensate only A_OFFSET_LIMIT angle.
    A_OFFSET_LIMIT,
    /// Horizontal smoothing coefficient of constant camera movement. The range
    /// of values depends on the specific implementation of the stibilisation
    /// algorithm. Default values [0-1]: 0 - the library will not compensate for
    /// constant camera motion, video will not be stabilized, 1 - no smoothing
    /// of constant camera motion (the library will compensate for the current
    /// picture drift completely without considering constant motion).
    X_FILTER_COEFF,
    /// Vertical smoothing coefficient of constant camera movement. The range
    /// of values depends on the specific implementation of the stibilisation
    /// algorithm. Default values [0-1]: 0 - the library will not compensate for
    /// constant camera motion, video will not be stabilized, 1 - no smoothing
    /// of constant camera motion (the library will compensate for the current
    /// picture drift completely without considering constant motion).
    Y_FILTER_COEFF,
    /// Rotational smoothing coefficient of constant camera movement. The range
    /// of values depends on the specific implementation of the stibilisation
    /// algorithm. Default values [0-1]: 0 - the library will not compensate for
    /// constant camera motion, video will not be stabilized, 1 - no smoothing
    /// of constant camera motion (the library will compensate for the current
    /// picture drift completely without considering constant motion).
    A_FILTER_COEFF,
    /// Stabilisation mode:
    /// 0 - Stabilisation off. The library should just copy input image.
    /// 1 - Stabilisation on.
    MODE,
    /// Transparent border mode:
    /// 0 - Not transparent borders (black borders).
    /// 1 - Transparent borders (parts of previous images).
    /// Particular implementation can have additional modes.
    TRANSPARENT_BORDER,
    /// Constant horizontal image offset in pixels. The library should add this
    /// offset to each processed video frame.
    CONST_X_OFFSET,
    /// Constant vertical image offset in pixels. The library should add this
    /// offset to each processed video frame.
    CONST_Y_OFFSET,
    /// Constant rotational angle in radians. The library should add this
    /// offset to each processed video frame.
    CONST_A_OFFSET,
    /// Instant (for one frame) horizontal image offset in pixels. The library
    /// should add this offset to next processed video frame.
    INSTANT_X_OFFSET,
    /// Instant (for one frame) vertical image offset in pixels. The library
    /// should add this offset to next processed video frame.
    INSTANT_Y_OFFSET,
    /// Instant (for one frame) rotational angle in radians. The library
    /// should add this offset to next processed video frame.
    INSTANT_A_OFFSET,
    /// Algorithm type. Default values:
    /// 0 - 2D type 1. Stabilisation only on horizonatal and vertical.
    /// 1 - 2D type 2. Stabilisation only on horizonatal and vertical.
    /// 2 - 3D. Stabilisation on horizontal and vertical + rotation.
    /// Particular implementation can have unique values.
    TYPE,
    /// Cat frequency, Hz. Stabiliser will block vibrations with frequency
    /// > CUT_FREQUENCY_HZ.
    CUT_FREQUENCY_HZ,
    /// Frames per second of input video.
    FPS,
    /// Processing time, mks. Processing time for last video frame.
    PROCESSING_TIME_MKS,
    /// Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal,
    /// 3 - File and terminal.
    LOG_MODE
};
```

Table 4 - Video stabiliser parameters description supported by VStabiliserOpenCv.

Parameter	Description
SCALE_FACTOR	Scale factor. Value: If 1 the library will process original frame size, if 2 the library will scale original frame size by 2, if 3 - by 3. VStabiliserOpenCv class supports 2D FFT algorithm which scales input image to wise 512x512 or 256x256 depends on scale factor. To chose particular size (512x512 or 256x256) the library devide input frame height by scale factor. If result >= 512 the library will use 512x512 size for internal algorithms. Otherwise the library will use 256x256 size.
X_OFFSET_LIMIT	Maximum horizontal image shift in pixels per video frame. If image shift bigger than this limit the library will compensate only X_OFFSET_LIMIT shift.
Y_OFFSET_LIMIT	Maximum vertical image shift in pixels per video frame. If image shift bigger than this limit the library will compensate only Y_OFFSET_LIMIT shift.
A_OFFSET_LIMIT	Maximum rotational image angle in radians per video frame. If image absolute rotational angle bigger than this limit the library will compensate only A_OFFSET_LIMIT angle.
X_FILTER_COEFF	Horizontal smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
Y_FILTER_COEFF	Vertical smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
A_FILTER_COEFF	Rotational smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
MODE	Stabilisation mode: 0 - Stabilisation off. The library just copy input image. 1 - Stabilisation on.
TRANSPARENT_BORDER	Transparent border mode: 0 - Not transparent borders (black borders). 1 - Transparent borders (parts of previous images).
CONST_X_OFFSET	Constant horizontal image offset in pixels. The library will add this offset to each processed video frame. This values used for boresight correction.

Parameter	Description
CONST_Y_OFFSET	Constant vertical image offset in pixels. The library will add this offset to each processed video frame. This values used for boresight correction.
CONST_A_OFFSET	Constant rotational angle in radians. The library will add this offset to each processed video frame. This values used for boresight correction.
INSTANT_X_OFFSET	Read only. Value of horizontal offset (pixels) implemented to last processed video frame.
INSTANT_Y_OFFSET	Read only. Value of vertical offset (pixels) implemented to last processed video frame.
INSTANT_A_OFFSET	Read only. Value of rotation angle (radians) implemented to last processed video frame.
TYPE	Three types of stabilisation algorithm are implemented in the library: **0 - 2D based on FFT**. Fastest algorithm but only for 2D stabilisation. Works stable for low light conditions and for low contrast images. **1 - 2D based on optical flow**. Gives good accuracy but lower speed as 2D FFT and requires contrast objects on video. **2 - 3D based on optical flow**. Gives best accuracy but lower speed as 2D FFT and requires contrast objects on video.
CUT_FREQUENCY_HZ	Not supported by VStabiliserOpenCv.
FPS	Not supported by VStabiliserOpenCv.
PROCESSING_TIME_MKS (Read only parameter)	Processing time, mks. Processing time for last video frame.
LOG_MODE	Not supported by VStabiliserOpenCv.

VStabiliserCommand enum

VStabiliserCommand enum lists video stabiliser commands. **VStabiliserCommand** enum declared in **VStabiliser.h** file of the VStabiliser library. Enum declaration.

```
enum class VStabiliserCommand
{
    /// Reset stabilisation algorithm.
    RESET = 1,
    /// Enable stabilisation. After execution parameter MODE must be set to 1.
    ON,
    /// Disable stabilisation. After execution parameter MODE must be set to 0.
    OFF
};
```

Table 3 - Video stabiliser commands description.

Parameter	Description
RESET	Reset stabilisation algorithm.
ON	Enable stabilisation. After execution parameter **MODE** must be set to 1.
OFF	Disable stabilisation. After execution parameter **MODE** must be set to 0.

VStabiliserParams class description

VStabiliserParams class used for video stabiliser initialization (**initVStabiliser(...)** method) or to get all actual params (**getParams()** method). Also **VStabiliserParams** provide structure to write/read params from JSON files (**JSON_READABLE** macro) and provide methos to encode and decode params. **VStabiliserParams** class declared in **VStabiliser.h** file of the VStabiliser library.

VStabiliserParams class declaration

VStabiliser.h file contains **VStabiliserParams** class declaration. **VStabiliserParams** class declared in **VStabiliser.h** file of the VStabiliser library. Class declaration:

```
class VStabiliserParams
{
public:
    /// Scale factor. Value depends on implementation. Default:
    /// If 1 the library will process original frame size, if 2
    /// the library will scale original frane size by 2, if 3 - by 3.
    int scaleFactor{1};
    /// Maximum horizontal image shift in pixels per video frame. If image shift
    /// bigger than this limit, the library should compensate only xOffsetLimit
    /// shift.
    int xOffsetLimit{150};
    /// Maximum vertical image shift in pixels per video frame. If image shift
    /// bigger than this limit, the library should compensate only yOffsetLimit
    /// shift.
    int yOffsetLimit{150};
    /// Maximum rotational image angle in radians per video frame. If image
    /// absolute rotational angle bigger than this limit, the library should
    /// compensate only aOffsetLimit angle.
    float aOffsetLimit{10.0f};
    /// Horizontal smoothing coefficient of constant camera movement. The range
    /// of values depends on the specific implementation of the stibilisation
    /// algorithm. Default values [0-1]: 0 - the library will not compensate for
    /// constant camera motion, video will not be stabilized, 1 - no smoothing
    /// of constant camera motion (the library will compensate for the current
    /// picture drift completely without considering constant motion).
    float xFilterCoeff{0.9f};
    /// Vertical smoothing coefficient of constant camera movement. The range
    /// of values depends on the specific implementation of the stibilisation
    /// algorithm. Default values [0-1]: 0 - the library will not compensate for
    /// constant camera motion, video will not be stabilized, 1 - no smoothing
    /// of constant camera motion (the library will compensate for the current
    /// picture drift completely without considering constant motion).
    float yFilterCoeff{0.9f};
    /// Rotational smoothing coefficient of constant camera movement. The range
    /// of values depends on the specific implementation of the stibilisation
    /// algorithm. Default values [0-1]: 0 - the library will not compensate for
    /// constant camera motion, video will not be stabilized, 1 - no smoothing
    /// of constant camera motion (the library will compensate for the current
    /// picture drift completely without considering constant motion).
    float aFilterCoeff{0.9f};
    /// Enable/disable stabilisation.
    bool enable{true};
    /// Enable/disable transparent borders.
    bool transparentBorder{true};
    /// Constant horizontal image offset in pixels. The library should add this
    /// offset to each processed video frame.
    int constXOffset{0};
    /// Constant vertical image offset in pixels. The library should add this
    /// offset to each processed video frame.
    int constYOffset{0};
    /// Constant rotational angle in radians. The library should add this
    /// offset to each processed video frame.
    float constAOffset{0.0f};
    /// Instant (for one frame) horizontal image offset in pixels. The library
    /// should add this offset to next processed video frame.
    int instantXOffset{0};
    /// Instant (for one frame) vertical image offset in pixels. The library
    /// should add this offset to next processed video frame.
    int instantYOffset{0};
    /// Instant (for one frame) rotational angle in radians. The library
    /// should add this offset to next processed video frame.
    float instantAOffset{0.0f};
    /// Algorithm type. Default values:
    /// 0 - 2D type 1. Stabilisation only on horizonatal and vertical.
    /// 1 - 2D type 2. Stabilisation only on horizonatal and vertical.
    /// 2 - 3D. Stabilisation on horizontal and vertical + rotation.
    /// Particular implementation can have unique values.
    int type{2};
    /// Cat frequency, Hz. Stabiliser will block vibrations with frequency
    /// greater than cutFrequencyHz.
    float cutFrequencyHz{2.0f};
    /// Frames per second of input video.
    float fps{30.0f};
    /// Processing time, mks. Processing time for last video frame.
    int processingTimeMks{0};
    /// Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal,
    /// 3 - File and terminal.
    int logMod{0};

    JSON_READABLE(VStabiliserParams, scaleFactor, xOffsetLimit, yOffsetLimit,
                  aOffsetLimit, xFilterCoeff, yFilterCoeff, aFilterCoeff,
                  enable, transparentBorder, constXOffset, constYOffset,
                  constAOffset, type, cutFrequencyHz, fps, logMod);

    /**
```

```
     * @brief operator =
     * @param src Source object.
     * @return VStabiliserParams obect.
     */
    VStabiliserParams& operator= (const VStabiliserParams& src);

    /**
     * @brief Encode params.
     * @param data Pointer to data buffer.
     * @param size Size of data.
     * @param mask Pointer to params mask to include in data.
     */
    void encode(uint8_t* data, int& size, VStabiliserParamsMask* mask = nullptr);

    /**
     * @brief Decode params.
     * @param data Pointer to data.
     * @return TRUE is params decoded or FALSE if not.
     */
    bool decode(uint8_t* data);
};
```

Table 5 - VStabiliserParams class fields description is equivalent to **VStabiliserParam** enum description.

Field	type	Description
scaleFactor	int	Scale factor. Value: If 1 the library will process original frame size, if 2 the library will scale original frame size by 2, if 3 - by 3. VStabiliserOpenCv class supports 2D FFT algorithm which scales input image to wise 512x512 or 256x256 depends on scale factor. To chose particular size (512x512 or 256x256) the library devide input frame height by scale factor. If result >= 512 the library will use 512x512 size for internal algorithms. Otherwise the library will use 256x256 size.
xOffsetLimit	int	Maximum horizontal image shift in pixels per video frame. If image shift bigger than this limit the library will compensate only xOffsetLimit shift.
yOffsetLimit	int	Maximum vertical image shift in pixels per video frame. If image shift bigger than this limit the library will compensate only yOffsetLimit shift.
aOffsetLimit	float	Maximum rotational image angle in radians per video frame. If image absolute rotational angle bigger than this limit the library will compensate only aOffsetLimit angle.

Field	type	Description
xFilterCoeff	float	Horizontal smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
yFilterCoeff	float	Vertical smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
aFilterCoeff	float	Rotational smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
enable	bool	Enable/disable stabilisation.
transparentBorder	bool	Enable/disable transparent borders.
constXOffset	int	Constant horizontal image offset in pixels. The library will add this offset to each processed video frame. This values used for boresight correction.
constYOffset	int	Constant vertical image offset in pixels. The library will add this offset to each processed video frame. This values used for boresight correction.
constAOffset	float	Constant rotational angle in radians. The library will add this offset to each processed video frame. This values used for boresight correction.
instantXOffset	int	Value of horizontal offset (pixels) implemented to last processed video frame.
instantYOffset	int	Value of vertical offset (pixels) implemented to last processed video frame.
instantAOffset	int	Value of rotation angle (radians) implemented to last processed video frame.

Field	type	Description
type	int	Three types of stabilisation algorithm are implemented in the library: **0 - 2D based on FFT**. Fastest algorithm but only for 2D stabilisation. Works stable for low light conditions and for low contrast images. **1 - 2D based on optical flow**. Gives good accuracy but lower speed as 2D FFT and requires contrast objects on video. **2 - 3D based on optical flow**. Gives best accuracy but lower speed as 2D FFT and requires contrast objects on video.
cutFrequencyHz	float	Not supported by VStabiliserOpenCv.
fps	float	Not supported by VStabiliserOpenCv.
processingTimeMks (read only parameter)	int	Processing time, mks. Processing time for last video frame.
logMod	int	Not supported by VStabiliserOpenCv.

None: *VStabiliserParams class fiellds listed in Table 4 reflects params set/get by methods setParam(...) and getParam(...).*

Serialize video stabiliser params

VStabiliserParams class provides method **encode(...)** to serialize video stabiliser params (fields of VStabiliserParams class, see Table 4). Serialization of params necessary in case when you need to send params via communication channels. Method provide options to exclude particular parameters from serialization. To do this method inserts binary mask (3 bytes) where each bit represents particular parameter and **decode(...)** method recognizes it. Method declaration:

```
void encode(uint8_t* data, int& size, VStabiliserParamsMask* mask = nullptr);
```

Parameter	Value
data	Pointer to data buffer.
size	Size of encoded data.
mask	Parameters mask - pointer to **VStabiliserParamsMask** structure. **VStabiliserParamsMask** (declared in VStabiliser.h file) determines flags for each field (parameter) declared in **VStabiliserParams** class. If the user wants to exclude any parameters from serialization, he can put a pointer to the mask. If the user wants to exclude a particular parameter from serialization, he should set the corresponding flag in the VStabiliserParamsMask structure.

VStabiliserParamsMask structure declaration:

```
typedef struct VStabiliserParamsMask
{
    bool scaleFactor{true};
    bool xOffsetLimit{true};
    bool yOffsetLimit{true};
    bool aOffsetLimit{true};
    bool xFilterCoeff{true};
    bool yFilterCoeff{true};
    bool aFilterCoeff{true};
    bool enable{true};
    bool transparentBorder{true};
    bool constXOffset{true};
    bool constYOffset{true};
    bool constAOffset{true};
    bool instantXOffset{false};
    bool instantYOffset{false};
    bool instantAOffset{false};
    bool type{true};
    bool cutFrequencyHz{true};
    bool fps{true};
    bool processingTimeMks{true};
    bool logMod{true};
} VStabiliserParamsMask;
```

Example without parameters mask:

```
// Encode data.
VStabiliserParams in;
uint8_t data[1024];
int size = 0;
in.encode(data, size);
cout << "Encoded data size: " << size << " bytes" << endl;
```

Example without parameters mask:

```
// Prepare mask.
VStabiliserParamsMask mask;
mask.scaleFactor = true; // Exclude scaleFactor. Others by default.

// Encode data.
VStabiliserParams in;
uint8_t data[1024];
int size = 0;
in.encode(data, size, &mask);
cout << "Encoded data size: " << size << " bytes" << endl;
```

Deserialize video stabiliser params

VStabiliserParams class provides method **decode(...)** to deserialize params (fields of VStabiliserParams class, see Table 4). Deserialization of params necessary in case when you need to receive params via communication channels. Method automatically recognizes which parameters were serialized by **encode(...)** method. Method declaration:

```
bool decode(uint8_t* data);
```

Parameter	Value
data	Pointer to data buffer.

Returns: TRUE if data decoded (deserialized) or FALSE if not.

Example:

```
// Encode data.
VStabiliserParams in;
uint8_t data[1024];
int size = 0;
in.encode(data, size);
cout << "Encoded data size: " << size << " bytes" << endl;

// Decode data.
VStabiliserParams out;
if (!out.decode(data))
    cout << "Can't decode data" << endl;
```

Read params from JSON file and write to JSON file

VStabiliser interface class library depends on **ConfigReader** library which provides method to read params from JSON file and to write params to JSON file. Example of writing and reading params to JSON file:

```
// Prepare random params.
VStabiliserParams in;
in.scaleFactor = rand() % 255;
in.xOffsetLimit = rand() % 255;
in.yOffsetLimit = rand() % 255;
in.aOffsetLimit = static_cast<float>(rand() % 255);

// Write params to file.
cr::utils::ConfigReader inConfig;
inConfig.set(in, "VStabiliserParams");
inConfig.writeToFile("TestVStabiliserParams.json");

// Read params from file.
cr::utils::ConfigReader outConfig;
if(!outConfig.readFromFile("TestVStabiliserParams.json"))
{
    cout << "Can't open config file" << endl;
    return false;
}

VStabiliserParams out;
if(!outConfig.get(out, "VStabiliserParams"))
{
    cout << "Can't read params from file" << endl;
    return false;
}
```

TestVStabiliserParams.json will look like:

```json
{
    "vStabiliserParams": {
        "aFilterCoeff": 5.0,
        "aOffsetLimit": 13.0,
        "constAOffset": 16.0,
        "constXOffset": 57,
        "constYOffset": 33,
        "cutFrequencyHz": 161.0,
        "enable": false,
        "fps": 90.0,
        "logMod": 99,
        "scaleFactor": 53,
        "transparentBorder": true,
        "type": 208,
        "xFilterCoeff": 76.0,
        "xOffsetLimit": 51,
        "yFilterCoeff": 230.0,
        "yOffsetLimit": 244
    }
}
```

Build and connect to your project

Typical commands to build **VStabiliserOpenCv** library:

```bash
sudo apt-get install libopencv-dev
cd VStabiliserOpenCv
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
```

If you want to connect **VStabiliserOpenCv** library to your CMake project as source code, you can do the following. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

You can add repository **VStabiliserOpenCv** as git submodule by commands (only if you have access to GitHub repository):

```
cd <your respository folder>
git submodule add https://github.com/ConstantRobotics-Ltd/VStabiliserOpenCv.git
3rdparty/VStabiliserOpenCv
git submodule update --init --recursive
```

In your repository folder, a new **3rdparty/VStabiliser** folder will be created, which contains files from **VStabiliser** repository along with its subrepositories **Frame** and **ConfigReader**. If you don't have access to GitHub repository, copy **VStabiliserOpenCv** repository folder to **3rdparty** folder to your repository. The new structure of your repository will be as follows:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    VStabiliserOpenCv
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should be containing:

```
cmake_minimum_required(VERSION 3.13)

################################################################################
## 3RD-PARTY
## dependencies for the project
################################################################################
project(3rdparty LANGUAGES CXX)

################################################################################
## SETTINGS
## basic 3rd-party settings before use
################################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

################################################################################
## CONFIGURATION
## 3rd-party submodules configuration
################################################################################
SET(${PARENT}_SUBMODULE_VSTABILISER_OPENCV              ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VSTABILISER_OPENCV)
    SET(${PARENT}_VSTABILISER_OPENCV                    ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VSTABILISER_OPENCV_EXAMPLES           OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VSTABILISER_OPENCV_BENCHMARK          OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VSTABILISER_OPENCV_DEMO_APP           OFF CACHE BOOL "" FORCE)
endif()

################################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
################################################################################
if (${PARENT}_SUBMODULE_VSTABILISER_OPENCV)
```

```
    add_subdirectory(VStabiliserOpenCv)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **VStabiliserOpenCv** to your project and excludes test applications and examples from compiling. The new structure of your repository will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    VStabiliserOpenCv
```

Next, you need to include the '3rdparty' folder in the main **CMakeLists.txt** file of your repository. Add the following string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next, you have to include VStabiliserOpenCv library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} VStabiliserOpenCv)
```

Done!

Simple example

Below is the code for a simple application that performs video stabilization in BGR24 format. An example library for other supported pixel formats is provided along with the application. The application uses the OpenCV library to capture and display video.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <VStabiliserOpenCv.h>

// Entry point.
int main(void)
{
    // Init video source.
    cv::VideoCapture cap;
    if (!cap.open("test.mp4"))
        return -1;

    // Get video frame size.
    int width = (int)cap.get(cv::CAP_PROP_FRAME_WIDTH);
    int height = (int)cap.get(cv::CAP_PROP_FRAME_HEIGHT);

    // Init images.
    cv::Mat opencvSrcFrame(cv::Size(width, height), CV_8UC3);
    cv::Mat opencvDstFrame(cv::Size(width, height), CV_8UC3);
    cr::video::Frame srcFrame(width, height, cr::video::Fourcc::BGR24);
    cr::video::Frame dstFrame(width, height, cr::video::Fourcc::BGR24);

    // Create video stabilizer object.
    cr::vstab::VStabiliserOpenCv videoStabilizer;

    // Set video stabilizer parameters.
    videoStabilizer.setParam(cr::vstab::VStabiliserParam::SCALE_FACTOR, 1);
    videoStabilizer.setParam(cr::vstab::VStabiliserParam::TYPE, 0);

    // Main loop.
    while (true)
    {
        // Capture next video frame.
        cap >> opencvSrcFrame;
        if (opencvSrcFrame.empty())
        {
            // If we have video we can set initial video position.
            cap.set(cv::CAP_PROP_POS_FRAMES, 1);
            // Reset video stabiliser.
            videoStabilizer.executeCommand(cr::vstab::VStabiliserCommand::RESET);
            continue;
        }

        // Copy video frame data from OpenCV image to Frame object.
        memcpy(srcFrame.data, opencvSrcFrame.data, srcFrame.size);

        // Stabilise frame.
        if (!videoStabilizer.stabilise(srcFrame, dstFrame))
            std::cout << "Stabilisation not calculated" << std::endl;

        // Copy Frame object data to OpenCV image.
        memcpy(opencvDstFrame.data, dstFrame.data, dstFrame.size);

        // Show source and result images.
        cv::imshow("SOURCE VIDEO", opencvSrcFrame);
        cv::imshow("RESULT VIDEO", opencvDstFrame);

        // Process keyboard events.
        switch (cv::waitKey(1))
        {
        case 27: // ESC - exit.
            exit(0);
        case 32: // SPACE - reset video stabilizer.
            videoStabilizer.executeCommand(cr::vstab::VStabiliserCommand::RESET);
            break;
        }
    }

    return 1;
}
```