Dehazer C++ library

v2.0.0

Table of contents

Overview

Dehazer C++ library implements video dehazing algorithm based on histogram manipulations. The library utilizes different histogram manipulations to improve the contrast of the video frame and to reduce the effect of haze. The library is implemented in C++ (C++17 standard) and exclusively relies on OpenCV library (version 4.5.0 and higher). This library is suitable for various types of camera (daylight, SWIR, MWIR and LWIR) and it provides simple programming interface. Each instance of the **Dehazer** C++ class object performs frame-by-frame processing of a video data stream, processing each video frame independently. The library depends on open source VFilter library (provides interface as well as defines data structures for various video filters implementation, source code included, Apache 2.0 license) and OpenCV library (version 4.5.0 and higher). Additionally demo application depends on open source SimpleFileDialog (provides dialog to open files, source code included, Apache 2.0 license).

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	11.12.2023	First version based on histogram manipulation.
2.0.0	21.02.2024	- Library updated with VFilter interface. - Added CLAHE algorithm as second algorithm type.

Library files

The library supplied by source code only. The user would be 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 to include 3rdparty libraries.
    VFilter ------------------- Files of VFilter interface library.
src --------------------------- Folder with library source code.
    CMakeLists.txt ------------ CMake file.
    Dehazer.h ----------------- Main library header file.
    DehazerVersion.h ---------- Header file with library version.
    DehazerVersion.h.in ------- File for CMake to generate version header.
    Dehazer.cpp --------------- C++ implementation file.
demo -------------------------- Folder for demo application files.
    CMakeLists.txt ------------ CMake file for demo application.
    3rdaprty ------------------ Folder with 3rdparty libraries.
        CMakeLists.txt -------- CMake file to include 3rdparty libraries.
        SimpleFileDialog ------ File dialog service library folder.
```

```
    main.cpp ------------------ Source C++ file of demo application.
example ----------------------- Folder for simple example.
    CMakeLists.txt ------------ CMake file of example.
    main.cpp ------------------ Source C++ file of example.
benchmark --------------------- Folder with test application (benchmark).
    CMakeLists.txt ------------ CMake file of test application (benchmark).
    main.cpp ------------------ Source C++ file of test application.
```

Key features and capabilities

Table 2 - Key features and capabilities.

Parameter and feature	Description
Programming language	C++ (standard C++17). Depends on OpenCV library (version 4.5.0 and higher).
Supported OS	Compatible with any operating system that supports the C++ compiler (C++17 standard) and OpenCV library (version 4.5.0 and higher).
Algorithm type	Implemented two types of algorithms: histogram manipulation based on whole image area and histogram manipulation based on local pixel area (CLAHE).
Supported pixel formats	GRAY, YUV24, YUYV, UYVY, NV12, NV21, YV12, YU12. The library uses pixels intensity for video processing. If the pixel format of the image doesn't include intensity channel it should be converted to proper format before applying dehazing.
Maximum and minimum video frame size	The minimum size of video frames to be processed is 16x16 pixels, and the maximum size is 8192x8192 pixels. The size of the video frames to be processed has a significant impact on the computation speed.
Calculation speed	The processing time per video frame depends on the computing platform used. The processing time per video frame can be estimated with the benchmark application.

Supported pixel formats

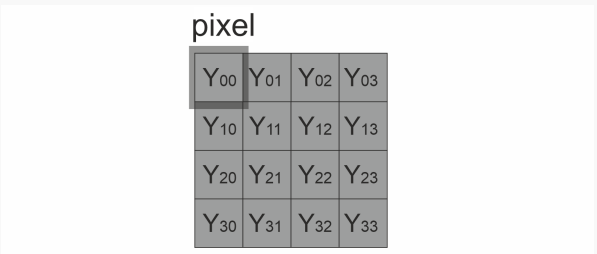
Frame library, which is included in **Dehazer** library contains **Fourcc** enum, which defines supported pixel formats (**Frame.h** file). **Dehazer** library supports intensity channel included pixel formats only (GRAY, YUV24, YUYV, UYVY, NV12, NV21, YV12, YU12). The library utilizes the intensity channel for video processing. **Fourcc** enum declaration:

```
enum class Fourcc
{
    /// RGB 24bit pixel format.
    RGB24 = MAKE_FOURCC_CODE('R', 'G', 'B', '3'),
```

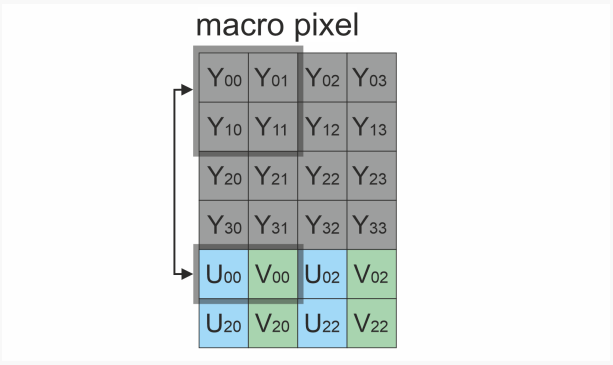
```
    /// BGR 24bit pixel format.
    BGR24 = MAKE_FOURCC_CODE('B', 'G', 'R', '3'),
    /// YUYV 16bits per pixel format.
    YUYV  = MAKE_FOURCC_CODE('Y', 'U', 'Y', 'V'),
    /// UYVY 16bits per pixel format.
    UYVY  = MAKE_FOURCC_CODE('U', 'Y', 'V', 'Y'),
    /// Grayscale 8bit.
    GRAY  = MAKE_FOURCC_CODE('G', '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.

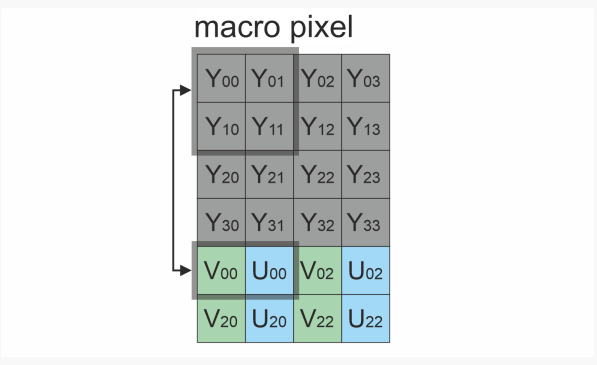
YUV24



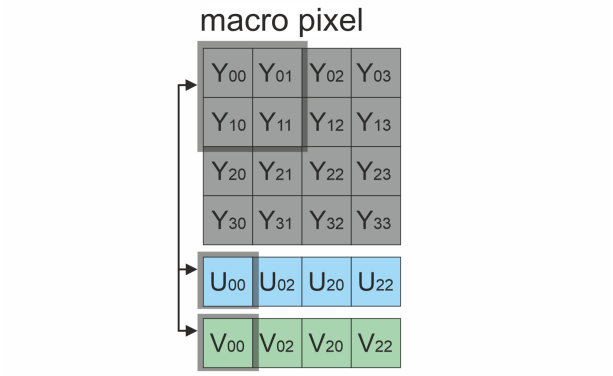
GRAY



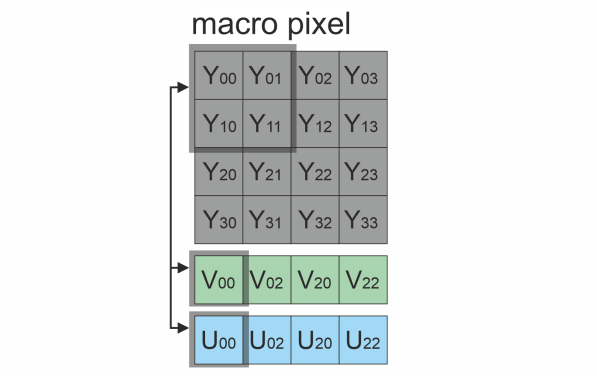
NV12



NV21



YU12



YV12

Library principles

The library is available as a source code only. To utilize the library as source code, developers have to incorporate the library files into their project. The usage sequence for the library is following:

1. Include the library files in the project.
2. Create an instance of the Dehazer C++ class. If you need multiple parallel cameras processing you have to create multiple Dehazer C++ class instances.
3. If necessary, modify the default library parameters using the setParam(...) method. Apply chosen type of the algorithm.
4. Create Frame class object for input frame.
5. Call the processFrame(...) method for frame processing.
6. The algorithm applies changes on the input original frame. Dehazing is not performed on selected pixels if detection mask was set.

Dehazer class description

Class declaration

Dehazer.h file contains **Dehazer** class declaration.

```
class Dehazer : public VFilter
{
public:

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

    //  Initialize video filter.
    bool initVFilter(VFilterParams& params) override;

    /// Set the value for a specific library parameter.
    bool setParam(VFilterParam id, float value) override;

    /// Get the value of a specific library parameter.
    float getParam(VFilterParam id) override;

    /// Get the structure containing all library parameters.
    void getParams(VFilterParams& params) override;

    /// Execute a VFilter action command.
    bool executeCommand(VFilterCommand id) override;

    /// Process frame. If mode == 0 (disabled) the method will skip frame.
    bool processFrame(cr::video::Frame& frame) override;

    // Set mask.
    bool setMask(cr::video::Frame mask) override;

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

getVersion method

The **getVersion()** method returns string of current version of **Dehazer** class. Method declaration:

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

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

```
std::cout << "Dehazer version: " << Dehazer::getVersion();
```

Console output:

```
Dehazer version: 2.0.0
```

initVFilter method

The **initVFilter(...)** method initializes dehazer filter with applied parameters. The method will can setParam(...) method for parameters: **mode**, **level** and **type**. Method declaration:

```
bool initVFilter(VFilterParams& params) override;
```

Parameter	Value
params	VFilterParams class object. Parameters supported by Dehazer: mode (default value 1) level (default value 0.0) type (type of algorithm, default value 0) If particular parameter is out of valid range, the library will set default values automatically.

Returns: TRUE if the dehazer initialized or FALSE if not.

setParam method

The **setParam(...)** method sets Dehazer parameter value. **setParam(...)** is a thread-safe method. This means that the **setParam(...)** method can be safely called from any thread. Method declaration:

```
bool setParam(VFilterParam id, float value) override;
```

Parameter	Description
id	Parameter ID according to VFilterParam enum. The library supports only parameters: **MODE**, **LEVEL** and **TYPE**. For other parameters the method will return FALSE.
value	Parameter value. Value depends on parameter ID.

Returns: TRUE if the parameter was set or FALSE if not (if it is read-only parameter or parameter value out of range).

getParam method

The **getParam(...)** method returns Dehazer parameter value. **getParam(...)** is a thread-safe method. This means that the **getParam(...)** method can be safely called from any thread. Method declaration:

```
float getParam(VFilterParam id) override;
```

Parameter	Description
id	Parameter ID according to VFilterParam enum. The library supports only parameters: **MODE**, **LEVEL** and **TYPE**. For other parameters the method will return **-1**.

Returns: parameter value or **-1**, if the parameter ID is not supported.

getParams method

The **getParams(...)** method is designed to obtain params structure. **VFilter** based library should provide thread-safe **getParams(...)** method call. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
params	Reference to VFilterParams object to store params.

executeCommand method

The **executeCommand(...)** method executes library action command. **VFilter** based library should provide thread-safe **executeCommand(...)** method call. This means that the **executeCommand(...)** method can be safely called from any thread. Method declaration:

```
bool executeCommand(VFilterCommand id) override;
```

Parameter	Description
id	Command ID according to VFilterCommand enum.

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

processFrame method

The **processFrame(..)** method processes the given frame. If the mode is set to 0 (disabled), the method will skip the frame and return true.

```
bool processFrame(cr::video::Frame& frame) override;
```

Parameter	Description
frame	Frame object to be processed. Implemented algorithm takes frame as a source and process original video frame. Dehazer processes only GRAY, YUV24, YUYV, UYVY, NV12, NV21, YV12, YU12 formats. The library uses pixels intensity for video processing. If the pixel format of the image doesn't include intensity channel it should be converted to proper format before dehazing.

Returns: TRUE if frame was successfully processed or skipped, FALSE if method encountered an error.

setMask method

The **setMask(...)** method designed to set filtration mask. The user can disable dehazing in any areas of the video frame. For this purpose the user can create an image of any size and configuration with GRAY pixel format. Mask image pixel values equal to 0 prohibit video filtration in the corresponding area of video frames. Any mask pixel value different than 0 allows to apply filter in this area. The method can be called either before video frame processing or during video frame processing. Method declaration:

```
bool setMask(cr::video::Frame mask) override;
```

Parameter	Description
mask	Filtration mask is see **Frame** object with GRAY pixel format. Dehazer omits image segments, where filtration mask pixel values equal 0. Mask can have any resolution. If resolution of mask (width and height) is not equal to video frame resolution, the library will scale this mask up to original processed video resolution.

Returns: TRUE if the the mask is accepted or FALSE if not (not valid pixel format or empty).

encodeSetParamCommand method of VFilter class

The **encodeSetParamCommand(...)** static method encodes command to change any **VFilter** parameter value. To control any video filter remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **VFilter** class contains static methods for encoding the control command. The **VFilter** 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, VFilterParam 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 VFilterParam enum.

Parameter	Description
value	Parameter value.

SET_PARAM command format:

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	Major	Major version of VFilter class.
2	Minor	Minor version of VFilter 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 **VFilter** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Random parameter value.
float outValue = static_cast<float>(rand() % 20);
// Encode command.
VFilter::encodeSetParamCommand(data, size, VFilterParam::LEVEL, outValue);
```

encodeCommand method of VFilter class

The **encodeCommand(...)** static method encodes command for **VFilter** remote control. To control any video filter remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **VFilter** class contains static methods for encoding the control command. The **VFilter** 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, VFilterCommand id);
```

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

COMMAND format:

Byte	Value	Description
0	0x00	COMMAND header value.
1	Major	Major version of VFilter class.
2	Minor	Minor version of VFilter 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 **VFilter** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[7];
// Size of encoded data.
int size = 0;
// Encode command.
VFilter::encodeCommand(data, size, VFilterCommand::RESTART);
```

decodeCommand method of VFilter class

The **decodeCommand(...)** static method decodes command on image filter side. Method declaration:

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

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM and 7 bytes for COMMAND.
paramId	VFilter parameter ID according to VFilterParam enum. After decoding SET_PARAM command the method will return parameter ID.

Parameter	Description
commandId	VFilter command ID according to VFilterCommand enum. After decoding COMMAND the method will return command ID.
value	VFilter 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.

decodeAndExecuteCommand method

decodeAndExecuteCommand(...) method decodes and executes command encoded on video filter side. The **decodeAndExecuteCommand(...)** method can be safely called from any thread. Method declaration:

```
bool decodeAndExecuteCommand(uint8_t* data, int size) override;
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Should be 11 bytes for SET_PARAM or 7 bytes for COMMAND.

Returns: TRUE if command decoded (SET_PARAM or COMMAND) and executed (action command or set param command) or FALSE if not.

Data structures

VFilterCommand enum

Enum declaration:

```
enum class VFilterCommand
{
    /// Restart image filter algorithm.
    RESET = 1,
    /// Enable filter.
    ON,
    /// Disable filter.
    OFF
};
```

Table 4 - Action commands description.

Command	Description
RESET	Not supported by **Dehazer** class.

Command	Description
ON	Enable video filter.
OFF	Disable video filter.

VFilterParam enum

Enum declaration:

```
enum class VFilterParam
{
    /// Current filter mode, usually 0 - off, 1 - on.
    MODE = 1,
    /// Enhancement level for particular filter, as a percentage in range from
    /// 0% to 100%.
    LEVEL,
    /// Processing time in microseconds. Read only parameter.
    PROCESSING_TIME_MCSEC,
    /// Type of the filter. Depends on the implementation.
    TYPE,
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    CUSTOM_1,
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    CUSTOM_2,
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    CUSTOM_3
};
```

Table 5 - Params description.

Parameter	Access	Description
MODE	read / write	Current filter mode, 0 - off, 1 - on. When the mode is set to 0, processFrame just forwards frame without processing and returns true.
LEVEL	read / write	Enhancement level applied in histogram filtering coefficient, as a percentage in range from 0% to 100%. The bigger value of level, the more influence on processing frame.
PROCESSING_TIME_MCSEC	read only	Processing time in microseconds. Read only parameter. Used to control performance of video filter.
TYPE	read / write	Type of the algorithm for dehazing. 0 - Custom histogram manipulation based on cumulative distribution function. 1 - CLAHE histogram equalization algorithm.

Parameter	Access	Description
CUSTOM_1	read / write	Custom parameter. Not supported.
CUSTOM_2	read / write	Custom parameter. Not supported.
CUSTOM_3	read / write	Custom parameter. Not supported.

VFilterParams class description

Class declaration

VFilterParams class is used to provide video filter parameters structure. Also **VFilterParams** provides possibility to write/read params from JSON files (**JSON_READABLE** macro) and provides methods to encode and decode params. **VFilterParams** interface class declared in **VFilter.h** file. Class declaration:

```
class VFilterParams
{
public:
    /// Current filter mode, usually 0 - off, 1 - on.
    int mode{ 0 };
    /// Enhancement level for particular filter, as a percentage in range from
    /// 0% to 100%.
    float level{ 0 };
    /// Processing time in microseconds. Read only parameter.
    int processingTimeMcSec{ 0 };
    /// Type of the filter. Depends on the implementation.
    int type{ 0 };
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    float custom1{ 0.0f };
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    float custom2{ 0.0f };
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    float custom3{ 0.0f };

    /// Macro from ConfigReader to make params readable/writable from JSON.
    JSON_READABLE(VFilterParams, mode, level, type, custom1, custom2, custom3)

    /// operator =
    VFilterParams& operator= (const VFilterParams& src);

    // Encode (serialize) params.
    bool encode(uint8_t* data, int bufferSize, int& size,
                                            VFilterParamsMask* mask = nullptr);
```

```
    // Decode (deserialize) params.
    bool decode(uint8_t* data, int dataSize);
};
```

Table 6 - **VFilterParams** class fields description is related to VFilterParam enum description.

Field	type	Description
mode	int	Current filter mode, 0 - off, 1 - on. When the mode is set to 0, processFrame just forwards frame without processing and returns true.
level	int	Enhancement level applied in histogram filtering coefficient, as a percentage in range from 0% to 100%. The bigger value of level, the more influence on processing frame.
processingTimeMcSec	int	Processing time in microseconds. Read only parameter. Used to control performance of video filter.
type	int	Type of the algorithm for dehazing. 0 - Custom histogram manipulation based on cumulative distribution function. 1 - CLAHE histogram equalization algorithm from **OpenCV** library.
custom1	float	Custom parameter. Not supported.
custom2	float	Custom parameter. Not supported.
custom3	float	Custom parameter. Not supported.

None: *VFilterParams class fields listed in Table 6* ***have to*** *reflect params set/get by methods setParam(...) and getParam(...).*

Serialize VFilter params

VFilterParams class provides method **encode(...)** to serialize VFilter params. Serialization of **VFilterParams** is necessary in case when image filter parameters have to be sent via communication channels. Method provides options to exclude particular parameters from serialization. To do this method inserts binary mask (1 byte) where each bit represents particular parameter and **decode(...)** method recognizes it. Method declaration:

```
bool encode(uint8_t* data, int bufferSize, int& size, VFilterParamsMask* mask = nullptr);
```

Parameter	Value
data	Pointer to data buffer. Buffer size must be >= 48 bytes.
bufferSize	Data buffer size. Buffer size must be >= 48 bytes.
size	Size of encoded data.

Parameter	Value
mask	Parameters mask - pointer to **VFilterParamsMask** structure. **VFilterParamsMask** (declared in VFilter.h file) determines flags for each field (parameter) declared in VFilterParams 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 **VFilterParamsMask** structure.

Returns: TRUE if params encoded (serialized) or FALSE if not.

VFilterParamsMask structure declaration:

```
struct VFilterParamsMask
{
    bool mode{ true };
    bool level{ true };
    bool processingTimeMcSec{ true };
    bool type{ true };
    bool custom1{ true };
    bool custom2{ true };
    bool custom3{ true };
};
```

Example without parameters mask:

```
// Prepare parameters.
cr::video::VFilterParams params;
params.level = 80.0f;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
params.encode(buffer, bufferSize, size);
```

Example with parameters mask:

```
// Prepare parameters.
cr::video::VFilterParams params;
params.level = 80.0;

// Prepare mask.
cr::video::VFilterParams mask;
// Exclude level.
mask.level = false;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
params1.encode(buffer, bufferSize, size, &mask);
```

Deserialize VFilter params

VFilterParams class provides method **decode(...)** to deserialize params. Deserialization of VFilterParams is necessary in case when it is needed to receive params via communication channels. Method automatically recognizes which parameters were serialized by **encode(...)** method. Method declaration:

```
bool decode(uint8_t* data, int dataSize);
```

Parameter	Value
data	Pointer to data buffer with serialized params.
dataSize	Size of command data.

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

Example:

```
// Prepare parameters.
cr::video::VFilterParams params1;
params1.level = 80;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
params1.encode(buffer, bufferSize, size);

// Decode (deserialize) params.
cr::video::VFilterParams params2;
params2.decode(buffer, size);
```

Read params from JSON file and write to JSON file

VFilter depends on open source 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:

```
// Write params to file.
cr::utils::ConfigReader inConfig;
cr::video::VFilterParams in;
inConfig.set(in, "vFilterParams");
inConfig.writeToFile("VFilterParams.json");

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

VFilterParams.json will look like:

```
{
    "vFilterParams":
    {
        "level": 50,
        "mode": 2,
        "type": 1,
        "custom1": 0.7f,
        "custom2": 12.0f,
        "custom3": 0.61f
    }
}
```

Build and connect to your project

Typical commands to build **Dehazer** library:

```
cd Dehazer
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want to connect **Dehazer** 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
```

Create folder 3rdparty in your repository folder and copy **Dehazer** repository folder there. The new structure of your repository will be as follows:

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

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(${PARENT}_SUBMODULE_DEHAZER                         ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_DEHAZER)
    SET(${PARENT}_DEHAZER                               ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_DEHAZER_DEMO                          OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_DEHAZER_EXAMPLE                       OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_DEHAZER_BENCHMARK                     OFF CACHE BOOL "" FORCE)
endif()

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

File **3rdparty/CMakeLists.txt** adds folder **Dehazer** 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
    Dehazer
```

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 **Dehazer** library in your **src/CMakeLists.txt** file:

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

Done!

Simple example

A simple application shows how to use the **Dehazer** library. The application opens a video file "test.mp4" and copies the video frame data into an object of the Frame class and performs dehazing.

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

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

    // Init dehazer.
    cr::video::Dehazer dehazer;

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

    // Init frames.
    cv::Mat srcOpenCvBgr(height, width, CV_8UC3);
    cv::Mat dstOpenCvBgr(height, width, CV_8UC3);
    cr::video::Frame frame(width, height, cr::video::Fourcc::YUV24);

    // Main loop.
    while (true)
```

```
    {
        // Capture next video frame.
        source >> srcOpenCvBgr;
        if (srcOpenCvBgr.empty())
        {
            source.set(cv::CAP_PROP_POS_FRAMES, 1);
            continue;
        }

        // Convert BGR to YUV.
        cv::Mat yuvImg(frame.height, frame.width, CV_8UC3, frame.data);
        cvtColor(srcOpenCvBgr, yuvImg, cv::COLOR_BGR2YUV);

        // Dehaze.
        dehazer.processFrame(frame);

        // Convert YUV to BGR.
        cvtColor(yuvImg, dstOpenCvBgr, cv::COLOR_YUV2BGR);

        // Show results.
        imshow("Result", dstOpenCvBgr);

        // Process keyboard events.
        if (cv::waitKey(1) == 27)
            exit(0);
    }

    return 1;
}
```