Motion Magnificator C++ library

v3.0.0

Table of contents

Overview

MotionMagnificator C++ library version is a versatile library that allows users to amplify subtle motion and temporal variations in digital video content. It allows camera system operators to detect moving objects that are not visible to the naked eye. Application area: perimeter security and drone detection. The library is implemented in C++ (C++17 standard). This library is suitable for various types of cameras (daylight, SWIR, MWIR and LWIR) and it provides robust magnification of movement for small objects. Each instance of the **MotionMagnificator** C++ class object performs frame-by-frame processing of a video data stream, processing each video frame independently. The library is designed only for not moving (or moving slowly) cameras or for PTZ cameras when observing in a certain sector. **The library can work real-time on low power CPU**. The library depends on open source VFilter library (provides interface as well as defines data structures for various video filters implementation, Apache 2.0 license). Additionally demo application depends on open source SimpleFileDialog (provides dialog to open files, Apache 2.0 license) and open source OpenCV library (provides functions to display video, Apache 2.0 license).

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	13.09.2021	First version.
2.0.0	23.10.2023	- New interface created. - New algorithm implemented.
2.0.1	13.11.2023	- Frame class updated.
2.0.2	28.12.2023	- Demo application updated. - Examples updated. - Benchmark added.
2.1.0	23.02.2024	- Interface update
3.0.0	27.02.2024	- Interface changed according to VFilter interface class.

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 of the library.
3rdparty ---------------------------- Folder with 3rdparty libraries.
    CMakeLists.txt ------------------- CMake file to include 3rdpary libraries.
    VFilter ------------------------- Files of VFilter interface library.
```

```
src ---------------------------------- Folder with library source code.
    CMakeLists.txt ------------------- CMake file.
    MotionMagnificator.h ------------- Main library header file.
    MotionMagnificatorVersion.h ------ Header file with library version.
    MotionMagnificatorVersion.h.in --- File for CMake to generate version header.
    MotionMagnificator.cpp ----------- C++ implementation file.
demo --------------------------------- Folder for demo application files.
    CMakeLists.txt ------------------- CMake file for demo application.
    3rdaprty ------------------------- Folder with 3rdparty libraries for demo
application.
        CMakeLists.txt --------------- CMake file to include 3rdpary libraries.
        SimpleFileDialog ------------- File dialog service library.
    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 benchmark.
    CMakeLists.txt ------------------- CMake file of benchmark.
    main.cpp ------------------------- Source C++ file of test app.
```

Key features and capabilities

Table 2 - Key features and capabilities.

Parameter and feature	Description
Programming language	C++ (standard C++17). No third-party dependencies.
Supported OS	Compatible with any operating system that supports the C++ compiler (C++17 standard).
Movement type	The library is able to magnify any kind of movement, but it is best suitable for tiny moving objects that are hard to spot with unarmed eye, e.g. incoming drone.
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 magnification.
Maximum and minimum video frame size	The minimum size of video frames to be processed is 32x32 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 demo application.

Parameter and feature	Description
Type of algorithm for movement magnification	A modified Lagrangian pyramid algorithm with temporal filtering and amplification was implemented. Amplification factor can be dynamically changed by user to serve different requirements.
Working conditions	Algorithm implemented in the library is designed to work on fixed cameras (or moving slowly) with any background conditions.

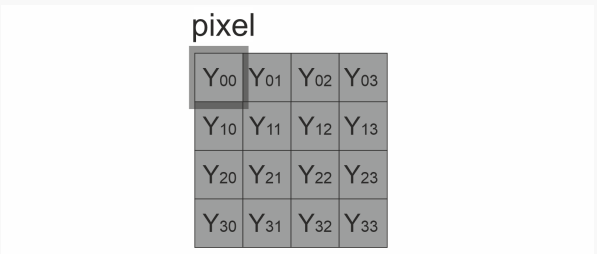
Supported pixel formats

The **Frame** library which included in **MotionMagnificator** library contains **Fourcc** enum which defines supported pixel formats (**Frame.h** file). **MotionMagnificator** library supports intensity channel included pixel formats only (GRAY, YUV24, YUYV, UYVY, NV12, NV21, YV12, YU12). The library uses 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.

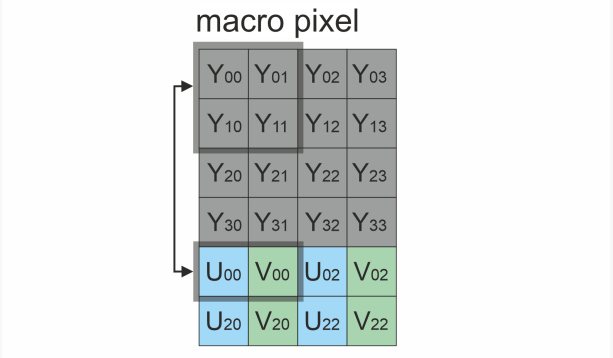
YUV24



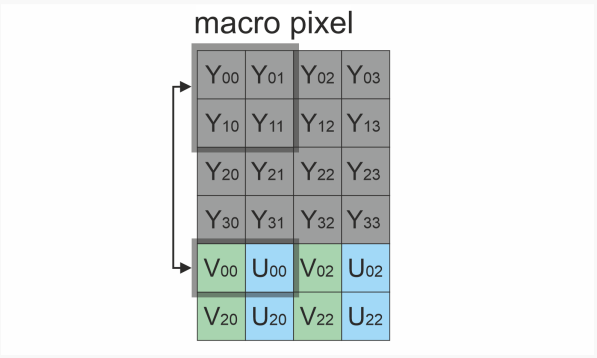
GRAY

YUYV

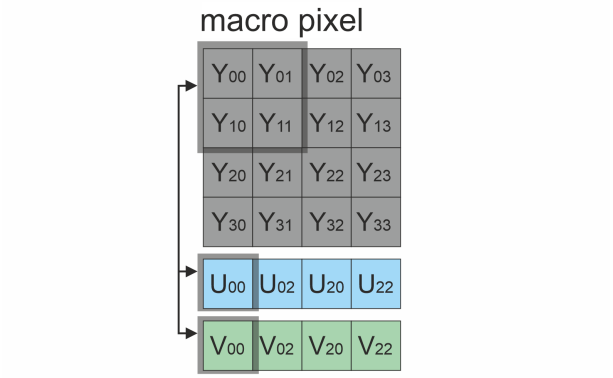
UYVY



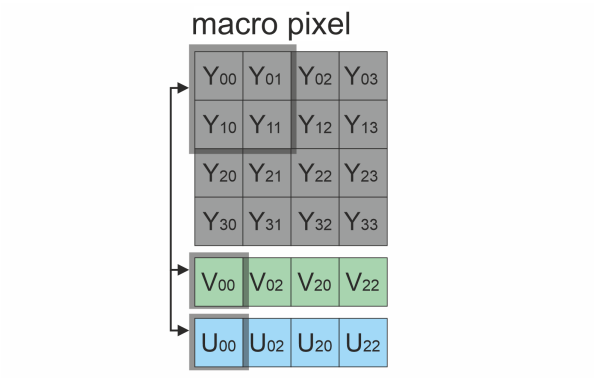
NV12



NV21



YU12



YV12

Library principles

The video motion magnification algorithm is implemented with multi-hypothesis support, incorporating temporal filtering and signal amplification. The algorithm involves the following sequential steps:

1. Acquiring the source video frame and retrieve intensity channel if needed.
2. Calculating pixel intensity values' differences.
3. Filtrating signal and applying amplification.
4. Creating result image by applying proper pixels' values to the intensity channel and then merged with incoming image.

The library is available as source code only. To utilize the library as source code, developers must incorporate the library files into their project. The usage sequence for the library is as follows:

1. Include the library files in the project.
2. Create an instance of the MotionMagnificator C++ class. If you need multiple parallel cameras processing you have to create multiple MotionMagnificator C++ class instances.
3. If necessary, modify the default library parameters using the setParam(...) method.
4. Create Frame class object for input frame.
5. Call the processFrame(...) method to magnify video data stream.

MotionMagnificator class description

MotionMagnificator class declaration

MotionMagnificator.h file contains **MotionMagnificator** class declaration:

```
class MotionMagnificator: public VFilter
{
public:

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

    /// Class constructor.
    MotionMagnificator();

    /// Class destructor.
    ~MotionMagnificator();

    /// Initialize motion magnificator.
    bool initVFilter(VFilterParams& params) override;

    /// Set motion magnificator param.
    bool setParam(VFilterParam id, float value) override;

    /// Get motion magnificator param value.
    float getParam(VFilterParam id) override;

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

    /// Process frame.
    bool processFrame(cr::video::Frame& frame) override;

    /// Set magnification 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 **MotionMagnificator** class. Method declaration:

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

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

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

Console output:

```
MotionMagnificator version: 3.0.0
```

initVFilter method

The **initVFilter(...)** method initializes **MotionMagnificator** class by VFilterParams object. Method uses only parameters: **mode** and **level** from VFilterParams class. Method declaration: Method declaration:

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

Parameter	Value
params	VFilterParams class object. Method uses only parameters: **mode** and **level** from VFilterParams class.

Returns: TRUE if the video filter initialized or FALSE if not.

setParam method

The **setParam(...)** method designed to set new **MotionMagnificator** object parameter value. **setParam(...)** is 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. Method supports only parameters: **MODE** and **LEVEL**. Other parameters will not be set.
value	Parameter value. Value depends on parameter ID.

Returns: TRUE if the parameter was set or FALSE if not.

getParam method

The **getParam(...)** method designed to obtain motion magnificator parameter value. **getParam(...)** is thread-safe method. This means that the **getParam(...)** method can be safely called from any thread. Method declaration:

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

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

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

executeCommand method

The **executeCommand(...)** method designed to execute motion magnificator command. **executeCommand(...)** is thread-safe method. 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 was executed or FALSE if not.

processFrame method

The **processFrame(...)** method designed to perform magnification algorithm. The library provides thread-safe **processFrame(...)** method call. This means that the **processFrame(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
frame	Video frame for processing. Motion magnificator 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 applying magnification.

Returns: TRUE if the video frame was processed FALSE if not. If motion magnificator disabled the method should return TRUE.

setMask method

The **setMask(...)** method designed to set magnification mask. The user can disable magnification in any areas of the video frame. For this purpose the user can create an image of any size and configuration with GRAY (preferable), NV12, NV21, YV12 or YU12 pixel format. Mask image pixel values equal to 0 prohibit motion magnification in the corresponding place of video frames. Any other mask pixel value other than 0 allows magnification at the corresponding location of video frames. The mask is used for motion magnification algorithms to compute a binary motion mask. 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	Image of magnification mask. Must have GRAY (preferable), NV12, NV21, YV12 or YU12 pixel format. The size and configuration of the mask image can be any. If the size of the mask image differs from the size of processed frames, the mask will be scaled by the library for processing.

Returns: TRUE if the the mask 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.
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.

Parameter	Description
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.
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

The **decodeAndExecuteCommand(...)** method decodes and executes command encoded by encodeSetParamCommand(...) and encodeCommand(...) methods on video filter side. The library provides thread-safe **decodeAndExecuteCommand(...)** method call. This means that 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. Must 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).

Data structures

VFilterCommand enum

Enum declaration:

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

Table 2 - Action commands description.

Command	Description
RESET	Reset image filter algorithm.
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 3 - Params description.

Parameter	Access	Description
MODE	read / write	Mode. Default: 0 - Off, 1 - On. If the magnificator is not activated, frame processing is not performed, it will be only forwarded.
LEVEL	read / write	Amplification factor adjusts the intensity of motion enhancement in videos, with higher values emphasizing motion changes and lower values maintaining the original motion characteristics. Supported values are from 0 to 100.
PROCESSING_TIME_MCSEC	read only	Processing time in microseconds. Read only parameter. Used to check performance of MotionMagnificator.
TYPE	read / write	**Not supported** by MotionMagnificator.
CUSTOM_1	read / write	**Not supported** by MotionMagnificator.
CUSTOM_2	read / write	**Not supported** by MotionMagnificator.
CUSTOM_3	read / write	**Not supported** by MotionMagnificator.

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

Field	type	Description
mode	int	Mode. Default: 0 - Off, 1 - On. If the magnificator is not activated, frame processing is not performed, it will be only forwarded.

Field	type	Description
level	float	Amplification factor adjusts the intensity of motion enhancement in videos, with higher values emphasizing motion changes and lower values maintaining the original motion characteristics. Supported values are from 0 to 100.
processingTimeMcSec	int	Processing time in microseconds. Read only parameter. Used to check performance of MotionMagnificator.
type	int	**Not supported** by MotionMagnificator.
custom1	float	**Not supported** by MotionMagnificator.
custom2	float	**Not supported** by MotionMagnificator.
custom3	float	**Not supported** by MotionMagnificator.

None: *VFilterParamsclass fields listed in Table 5* ***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.
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.

Parameter	Value
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 **MotionMagnificator** library:

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

If you want connect **MotionMagnificator** to your CMake project as source code you can make follow. For example, if your repository has structure:

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

Create folder **3rdparty** in your repository and copy **MotionMagnificator** repository folder there. New structure of your repository:

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

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

```
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(${PARENT}_SUBMODULE_MOTION_MAGNIFICATOR             ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_MOTION_MAGNIFICATOR)
    SET(${PARENT}_MOTION_MAGNIFICATOR                   ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_MOTION_MAGNIFICATOR_BENCHMARK         OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_MOTION_MAGNIFICATOR_DEMO_APP          OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_MOTION_MAGNIFICATOR_EXAMPLE           OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

Done!

Simple example

A simple application shows how to use the **MotionMagnificator** library. The application opens a video file "test.mp4".

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

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

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

    // Create frames.
    cv::Mat bgrImg;
    cr::video::Frame frameYuv(width, height, cr::video::Fourcc::YUV24);

    // Create motion magnificator object.
    cr::video::MotionMagnificator magnificator;

    // Main loop.
    while (true)
    {
        // Capture next video frame. Default BGR pixel format.
        videoSource >> bgrImg;
        if (bgrImg.empty())
        {
            // Set initial video position to replay.
            videoSource.set(cv::CAP_PROP_POS_FRAMES, 0);
            continue;
        }

        // Convert BGR to YUV for motion magnificator.
        cv::Mat yuvImg(height, width, CV_8UC3, frameYuv.data);
        cv::cvtColor(bgrImg, yuvImg, cv::COLOR_BGR2YUV);

        // Magnify movement with default params.
        magnificator.processFrame(frameYuv);

        // Conver result YUV to BGR to display.
        cv::cvtColor(yuvImg, bgrImg, cv::COLOR_YUV2BGR);

        // Show video.
        cv::imshow("VIDEO", bgrImg);
```

```
        // Wait ESC.
        if (cv::waitKey(1) == 27)
            return -1;
    }

    return 1;
}
```