DigitalZoom C++ library

v1.0.0

Table of contents

Overview

The **DigitalZoom** C++ library implements digital zoom based on OpenCV. The library supports different interpolation algorithms: bicubic interpolation, resampling using pixel area relation, Lanczos 8x8, bilinear interpolation, bit exact bilinear interpolation, nearest neighbor interpolation and bit exact nearest neighbor interpolation. The library supports digital zoom level from x1 to x20. The library depends on VFilter library (provides interface for video filter, source code included, Apache 2.0 license) and OpenCV library (version >= 4.5, linked, Apache 2.0 license). The library uses C++17 standard.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	25.02.2024	First version of the library.

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 third-party libraries.
    CMakeLists.txt ----------- CMake file which includes third-party libraries.
    VFilter ------------------ Source code of the VFilter interface library.
src -------------------------- Folder with source code of the library.
    CMakeLists.txt ----------- CMake file of the library.
    DigitalZoom.cpp ---------- Source code file of the library.
    DigitalZoom.h ------------ Header file which includes DigitalZoom class declaration.
    DigitalZoomVersion.h ----- Header file which includes version of the library.
    DigitalZoomVersion.h.in -- CMake service file to generate version file.
test ------------------------- Folder for test application.
    CMakeLists.txt ----------- CMake file for test application.
    main.cpp ----------------- Source code file of test application.
```

DigitalZoom class description

Class declaration

The **DigitalZoom** class declared in **DigitalZoom.h** file. Class declaration:

```
class DigitalZoom : public VFilter
{
public:

    /// Get the version of the DigitalZoom class.
    static std::string getVersion();

    /// Initialize digital zoom.
    bool initVFilter(VFilterParams& params) override;

    /// Set VFilter parameter.
    bool setParam(VFilterParam id, float value) override;

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

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

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

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

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

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

getVersion method

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

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

Method can be used without **DigitalZoom** class instance:

```
std::cout << "DigitalZoom version: " << cr::video::DigitalZoom::getVersion();
```

Console output:

```
DigitalZoom version: 1.0.0
```

initVFilter method

The **initVFilter(...)** method initializes digital zoom. Method uses only parameters: **mode**, **level** and **type** from VFilterParams class. Method declaration:

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

Parameter	Value
params	VFilterParams class object. Method uses only parameters: **mode**, **level** and **type** fields of VFilterParams class.

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

setParam method

The **setParam (...)** method sets new parameters value. The library provides thread-safe **setParam(...)** method call. 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**, **LEVEL** and **TYPE**. 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 returns parameter value. The library provides thread-safe **getParam(...)** method call. 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. Method supports only parameters: **MODE**, **LEVEL**, **TYPE** and **PROCESSING_TIME_MCSEC**. For other parameters the method will return **-1**.

Returns: parameter value or **-1** if the parameters doesn't exist (not supported in particular implementation).

getParams method

The **getParams(...)** method is designed to obtain all digital zoom params. 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(VFilterParams& params) override;
```

Parameter	Description
params	Reference to VFilterParams object to store params. Method will fill parameters: **mode**, **level**, **processingTimeMcSec** and **type**. Other parameters will be set to **-1.0**.

executeCommand method

The **executeCommand(...)** method executes video filter (digital zoom) action command. The library provides 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. Method supports only **ON** and **OFF** commands.

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

processFrame method

The **processFrame(...)** method designed to process frame (digital zoom). 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	Reference to Frame object.

Returns: TRUE if frame processed (digital zoom) or FALSE if not. If filter disabled the method will return TRUE without video frame processing.

setMask method

The **setMask(...)** method designed to set video filter mask. Mask **not supported by DigitalZoom** class. Method declaration:

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

Parameter	Description
mask	Filter mask is Frame object with GRAY pixel format. Mask **not supported by DigitalZoom** class.

Returns: TRUE in any cases.

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.

Byte	Value	Description
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.

Byte	Value	Description
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	**Not supported by DigitalZoom class**.
ON	Enable digital zoom.
OFF	Disable digital zoom.

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	Digital zoom mode: **0** - disabled, **1** - enabled.
LEVEL	read / write	Digital zoom level. Valid values from **1.0** to **20.0**.
PROCESSING_TIME_MCSEC	read only	Processing time in microseconds. Read only parameter. Used to check performance of digital zoom.
TYPE	read / write	Digital zoom interpolation time: **0** - [DEFAULT] Bicubic interpolation (cv::INTER_CUBIC in OpenCV). **1** - Resampling using pixel area relation (cv::INTER_AREA in OpenCV). **2** - Lanczos 8x8 (cv::INTER_LANCZOS4 in OpenCV). **3** - Bilinear interpolation (cv::INTER_LINEAR in OpenCV). **4** - Bit exact bilinear interpolation (cv::INTER_LINEAR_EXACT in OpenCV). **5** - Nearest neighbor interpolation (cv::INTER_NEAREST in OpenCV). **6** - Bit exact nearest neighbor interpolation (cv::INTER_NEAREST_EXACT in OpenCV).
CUSTOM_1	read / write	**Not supported by DigitalZoom class**.
CUSTOM_2	read / write	**Not supported by DigitalZoom class**.
CUSTOM_3	read / write	**Not supported by DigitalZoom class**.

VFilterParams class description

Class declaration

The **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 4 - **VFilterParams** class fields description is related to VFilterParam enum description.

Field	type	Description
mode	int	Digital zoom mode: **0** - disabled, **1** - enabled.
level	float	Digital zoom level. Valid values from **1.0** to **20.0**.

Field	type	Description
processingTimeMcSec	int	Processing time in microseconds. Read only parameter. Used to check performance of digital zoom.
type	int	Digital zoom interpolation time: **0** - [DEFAULT] Bicubic interpolation (cv::INTER_CUBIC in OpenCV). **1** - Resampling using pixel area relation (cv::INTER_AREA in OpenCV). **2** - Lanczos 8x8 (cv::INTER_LANCZOS4 in OpenCV). **3** - Bilinear interpolation (cv::INTER_LINEAR in OpenCV). **4** - Bit exact bilinear interpolation (cv::INTER_LINEAR_EXACT in OpenCV). **5** - Nearest neighbor interpolation (cv::INTER_NEAREST in OpenCV). **6** - Bit exact nearest neighbor interpolation (cv::INTER_NEAREST_EXACT in OpenCV).
custom1	float	VFilter custom parameter. Custom parameters used when particular image filter has specific unusual parameter.
custom2	float	VFilter custom parameter. Custom parameters used when particular image filter has specific unusual parameter.
custom3	float	VFilter custom parameter. Custom parameters used when particular image filter has specific unusual parameter.

None: *VFilterParams class fields listed in Table 4* ***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 **DigitalZoom**:

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

If you want connect **DigitalZoom** 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 **DigitalZoom** repository folder there. New structure of your repository:

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

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(${PARENT}_SUBMODULE_DIGITAL_ZOOM                    ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_DIGITAL_ZOOM)
    SET(${PARENT}_DIGITAL_ZOOM                          ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_DIGITAL_ZOOM_TEST                     OFF CACHE BOOL "" FORCE)
endif()

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

File **3rdparty/CMakeLists.txt** adds folder **DigitalZoom** to your project and excludes test application from compiling. Your repository new structure will be:

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

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

```
add_subdirectory(3rdparty)
```

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

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

Done!

Example

The example creates artificial image and provides user interface to control mode, level and type params:

```
#include <iostream>
#include <cstdint>
#include <string.h>
#include <opencv2/opencv.hpp>
#include "DigitalZoom.h"

int main(void)
{
    std::cout << "DigitalZoom v" <<
    cr::video::DigitalZoom::getVersion() << std::endl;

    // Prepare source artificial image.
    const int width = 1280;
    const int height = 720;
    cv::Mat srcImg = cv::Mat::zeros(height, width, CV_8UC3);
    cv::circle(srcImg, cv::Point(width / 2, height / 2),
               height / 2, cv::Scalar(0, 0, 255), cv::FILLED, cv::LINE_AA);
    cv::circle(srcImg, cv::Point(width / 2, height / 2),
               height / 3, cv::Scalar(0, 255, 0), cv::FILLED, cv::LINE_AA);
    cv::circle(srcImg, cv::Point(width / 2, height / 2),
               height / 4, cv::Scalar(255, 0, 0), cv::FILLED, cv::LINE_AA);
    cv::circle(srcImg, cv::Point(width / 2, height / 2),
               height / 5, cv::Scalar(0, 0, 255), cv::FILLED, cv::LINE_AA);
    cv::circle(srcImg, cv::Point(width / 2, height / 2),
               height / 7, cv::Scalar(0, 255, 0), cv::FILLED, cv::LINE_AA);
    cv::circle(srcImg, cv::Point(width / 2, height / 2),
               height / 8, cv::Scalar(255, 0, 0), cv::FILLED, cv::LINE_AA);
```

```
    cv::circle(srcImg, cv::Point(width / 2, height / 2),
               height / 9, cv::Scalar(0, 0, 255), cv::FILLED, cv::LINE_AA);
    cv::circle(srcImg, cv::Point(width / 2, height / 2),
               height / 10, cv::Scalar(0, 255, 0), cv::FILLED, cv::LINE_AA);
    cr::video::Frame frame(width, height, cr::video::Fourcc::BGR24);

    // Create digital zoom object and params.
    cr::video::DigitalZoom zoom;
    cr::video::VFilterParams params;

    // Main loop.
    while (true)
    {
        // Copy source image data for Frame object.
        memcpy(frame.data, srcImg.data, frame.size);

        // Digital zoom.
        zoom.processFrame(frame);

        // Get digital zoom params (VFilterParams).
        zoom.getParams(params);

        // Draw params.
        cv::Mat resultImg(height, width, CV_8UC3, frame.data);
        cv::putText(resultImg, "[1]Mode: " +
                    std::string(params.mode == 0 ? "OFF" : "ON"),
                    cv::Point(5, 25), cv::FONT_HERSHEY_SIMPLEX, 0.7,
                    cv::Scalar(255, 255, 0), 1, cv::LINE_AA);
        cv::putText(resultImg, "[2/3]Level: " + std::to_string(params.level),
                    cv::Point(5, 50), cv::FONT_HERSHEY_SIMPLEX, 0.7,
                    cv::Scalar(255, 255, 0), 1, cv::LINE_AA);
        std::string type;
        switch (params.type) {
        case 0: type = "INTER_CUBIC"; break;
        case 1: type = "INTER_AREA"; break;
        case 2: type = "INTER_LANCZOS4"; break;
        case 3: type = "INTER_LINEAR"; break;
        case 4: type = "INTER_LINEAR_EXACT"; break;
        case 5: type = "INTER_NEAREST"; break;
        case 6: type = "INTER_NEAREST_EXACT"; break;
        default: type = "INTER_CUBIC"; break; }
        cv::putText(resultImg, "[4/5]Type: " + type,
                    cv::Point(5, 75), cv::FONT_HERSHEY_SIMPLEX, 0.7,
                    cv::Scalar(255, 255, 0), 1, cv::LINE_AA);
        cv::putText(resultImg, "Time, mcsec: " +
                    std::to_string(params.processingTimeMcSec),
                    cv::Point(5, 100), cv::FONT_HERSHEY_SIMPLEX, 0.7,
                    cv::Scalar(255, 255, 0), 1, cv::LINE_AA);

        // Show results.
        cv::imshow("DigitalZoom", resultImg);

        // Process keyboard events.
        switch (cv::waitKey(30))
        {
        case 27:
```

```
            return 1;
        case 49:
            zoom.setParam(cr::video::VFilterParam::MODE, 1 - params.mode);
            break;
        case 50:
            zoom.setParam(cr::video::VFilterParam::LEVEL, params.level - 0.1f);
            break;
        case 51:
            zoom.setParam(cr::video::VFilterParam::LEVEL, params.level + 0.1f);
            break;
        case 52:
            zoom.setParam(cr::video::VFilterParam::TYPE, params.type - 1);
            break;
        case 53:
            zoom.setParam(cr::video::VFilterParam::TYPE, params.type + 1);
            break;
        }
    }
    return 1;
}
```