VSourceOpenCv C++ library

v1.0.0

Table of contents

Overview

VSourceOpenCv C++ library serves as an OpenCV wrapper for the **VSource** class. It provides an interface for handling video sources using the cv::VideoCapture instance. **VSourceOpenCv** supports **VSourceCommands** and most of the **VSourceParams**. Additionally, it incorporates logging capabilities from the **Logger** library, enabling efficient monitoring and debugging. VSourceOpenCv class dependency:
- **VSource** library provides standard interface as well defines data structures and rules for different video source classes.
- **Logger** library provides logging functions: printing in terminal and(or) printing in file.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	11.07.2023	First version

VSourceOpenCv class description

VSourceOpenCv class declaration

VSourceOpenCv interface class declared in **VSourceOpenCv.h** file. Class declaration:

```
class VSourceOpenCv : public VSource
{
public:

    /**
     * @brief Class constructor.
     */
    VSourceOpenCv();

    /**
     * @brief Class destructor.
     */
    ~VSourceOpenCv();

    /**
     * @brief Get string of current library version.
     * @return String of current library version.
     */
    static std::string getVersion();

    /**
     * @brief Open video source. All params will be set by default.
     * @param initString Init string. Format depends on implementation.
     * Default format: <video device or ID or file>;<width>;<height>
```

```
     * @return TRUE if the video source open or FALSE if not.
     */
    bool openVSource(std::string& initString) override;

    /**
     * @brief Init video source. All params will be set according
     * to structure.
     * @param params Video source parameters structure.
     * @return TRUE if the video source init or FALSE if not.
     */
    bool initVSource(VSourceParams& params) override;

    /**
     * @brief Get open status.
     * @return TRUE if video source open or FALSE if not.
     */
    bool isVSourceOpen() override;

    /**
     * @brief Close video source.
     */
    void closeVSource() override;

    /**
     * @brief Get new video frame.
     * @param frame Frame object to copy new data.
     * @param timeoutMsec Timeout to wait new frame data:
     * timeoutMsec == -1 - Method will wait endlessly until data arrive.
     * timeoutMsec == 0  - Method will only check if new data exist.
     * timeoutMsec > 0   - Method will wait new data specified time.
     * @return TRUE if new video frame exist and copied or FALSE if not.
     */
    bool getFrame(Frame& frame, int32_t timeoutMsec = 0) override;

    /**
     * @brief Set video source param.
     * @param id Parameter ID.
     * @param value Parameter value to set.
     * @return TRUE if property was set of FALSE.
     */
    bool setParam(VSourceParam id, float value) override;

    /**
     * @brief Get video source param value.
     * @param id Parameter ID.
     * @return Parameter value or -1.
     */
    float getParam(VSourceParam id) override;

    /**
     * @brief Get video source params structure.
     * @return Video source parameters structure.
     */
    VSourceParams getParams() override;
```

```
    /**
     * @brief Execute command.
     * @param id Command ID.
     * @return TRUE if the command accepted or FALSE if not.
     */
    bool executeCommand(VSourceCommand id) override;
};
```

getVersion method

getVersion() method returns string of current version of **VSourceOpencv** class. Particular video source class can have it's own **getVersion()** method. Method declaration:

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

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

```
std::cout << "VSourceOpencv class version: " << VSourceOpencv::getVersion() << std::endl;
```

Console output:

```
VSourceOpencv class version: 1.0.0
```

openVSource method

openVSource(...) method initializes OpenCv video source. Overrides the method from VSource class. Method declaration:

```
bool openVSource(std::string& initString) override;
```

Parameter	Value
initString	Initialization string. Format depends on implementation but it is recommended to keep default format: [video device or ID or file];[width];[height]. Example: "/dev/video0;1920;1080".

Returns: TRUE if the video source open or FALSE if not.

initVSource method

initVSource(...) method designed to initialize video source by set of parameters. Method declaration:

```
bool initVSource(VSourceParams& params) override;
```

Parameter	Value
params	VSourceParams structure (see **VSourceParams** class description). The video source should set parameters according to params structure. Particular video source can support not all parameters listed in VSourceParams class.

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

isVSourceOpen method

isVSourceOpen() method returns whether OpenCv video source is opened. Initialization status also included in **VSourceParams** class. Method declaration:

```
bool isVSourceOpen override;
```

Returns: TRUE if the video source open or FALSE if not.

closeVSource method

closeVSource() method intended to close video source. Method declaration:

```
void closeVSource() override;
```

getFrame method

getFrame(...) method intended to get input video frame. Video source should support auto reinitialization in case connection loss. Method declaration:

```
bool getFrame(Frame& frame, int32_t timeoutMsec = 0) override;
```

Parameter	Value
frame	Output video frame (see **Frame** class description). Always should be initialized with Fourcc::BGR24, if not it will be automatically changed.
timeoutMsec	Timeout to wait new frame data: - timeoutMs == -1 - Method will wait endlessly until new data arrive. - timeoutMs == 0 - Method will only check if new data exist. - timeoutMs > 0 - Method will wait new data specified time.

Returns: TRUE if new data exists and copied or FALSE if not.

setParam method

setParam(...) method designed to set new video source parameters value. Method declaration:

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

Parameter	Description
id	Video source parameter ID according to **VSourceParam** enum.
value	Video source parameter value.

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

getParam method

getParam(...) method designed to obtain video source parameter value. Method declaration:

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

Parameter	Description
id	Video source parameter ID according to **VSourceParam** enum.

Returns: parameter value or -1 if the parameters doesn't exist in particular video source class.

getParams method

getParams(...) method designed to obtain video source params structures. Method declaration:

```
VSourceParams getParams();
```

Returns: video source parameters structure (see **VSourceParams** class description).

executeCommand method

executeCommand(...) method designed to execute video source command. Method declaration:

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

Parameter	Description
id	Video source command ID according to **VSourceCommand** enum.

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

Data structures

VSourceOpenCv Class is based on **VSource** class and all its data structures: parameters (**VSourceParam** enum) and IDs for commands (**VSourceCommand** enum).

VSourceCommand enum

Enum declaration:

```
enum class VSourceCommand
{
    /// Restart.
    RESTART = 1
};
```

Table 2 - Video source commands description.

Command	Description
RESTART	Restart video source (close and open again).

VSourceParam enum

Enum declaration:

```
enum class VSourceParam
{
    /// [read/write] Logging mode. Values: 0 - Disable, 1 - Only file,
    /// 2 - Only terminal, 3 - File and terminal.
    LOG_LEVEL = 1,
    /// [read/write] Frame width. User can set frame width before initialization
    /// or after. Some video source classes may set width automatically.
    WIDTH,
    /// [read/write] Frame height. User can set frame height before
    /// initialization or after. Some video source classes may set height
    /// automatically.
    HEIGHT,
    /// [read/write] Gain mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    GAIN_MODE,
    /// [read/write] Gain value. Value: 0(min for particular video source class)
    /// - 65535(max for particular video source class).
    GAIN,
    /// [read/write] Exposure mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    EXPOSURE_MODE,
    /// [read/write] Exposure value. Value: 0(min for particular video source
    /// class) - 65535(max for particular video source class).
    EXPOSURE,
    /// [read/write] Focus mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
```

```
    FOCUS_MODE,
    /// [read/write] Focus position. Value: 0(full near) - 65535(full far).
    FOCUS_POS,
    /// [read only] Video capture cycle time. **VSource** class sets this value
    /// automatically. This parameter means time interval between two captured
    /// video frame.
    CYCLE_TIME_MKS,
    /// [read/write] FPS. User can set frame FPS before initialization or after.
    /// Some video source classes may set FPS automatically.
    FPS,
    /// [read only] Open flag. 0 - not open, 1 - open.
    IS_OPEN,
    /// [read/write] Custom parameter. Depends on implementation.
    CUSTOM_1,
    /// [read/write] Custom parameter. Depends on implementation.
    CUSTOM_2,
    /// [read/write] Custom parameter. Depends on implementation.
    CUSTOM_3
};
```

Table 3 - Video source params description. GAIN_MODE and custom params are not supported in **VSourceOpenCv** implementation.

Parameter	Access	Description
LOG_LEVEL	read / write	Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal, 3 - File and terminal.
WIDTH	read / write	Frame width. User can set frame width before initialization or after. Some video source classes may set width automatically.
HEIGHT	read / write	Frame height. User can set frame height before initialization or after. Some video source classes may set height automatically.
GAIN_MODE	read / write	Not supported.
GAIN	read / write	Gain value. Value: 0(min for particular video source class) - 65535(max for particular video source class).
EXPOSURE_MODE	read / write	Exposure mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.
EXPOSURE	read / write	Exposure value. Value: 0(min for particular video source class) - 65535(max for particular video source class).
FOCUS_MODE	read / write	Focus mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.
FOCUS_POS	read / write	Focus position. Value: 0(full near) - 65535(full far).
CYCLE_TIME_MKS	read only	Video capture cycle time. **VSourceOpenCv** class sets this value automatically depending on input FPS value. This parameter means time interval between two captured video frame.

Parameter	Access	Description
FPS	read / write	FPS. User can set frame FPS before initialization or after. Some video source classes may set FPS automatically.
IS_OPEN	read only	Open flag. 0 - not open, 1 - open.
CUSTOM_1	read / write	Not supported.
CUSTOM_2	read / write	Not supported.
CUSTOM_3	read / write	Not supported.

VSourceParams class description

VSourceParams class declaration

VSourceParams class used for video source initialization (**initVSource(...)** method) or to get all actual params (**getParams()** method). Also **VSourceParams** provide structure to write/read params from JSON files (**JSON_READABLE** macro, see **ConfigReader** class description) and provide methods to encode and decode params. Class declaration:

```
class VSourceParams
{
public:
    /// Logging mode. Values: 0 - Disable, 1 - Only file,
    /// 2 - Only terminal, 3 - File and terminal.
    int logLevel{0};
    /// Video source: file, video stream, video device, camera num, etc.
    std::string source{"/dev/video0"};
    /// FOURCC: RGB24, BGR24, YUYV, UYVY, GRAY, YUV24, NV12, NV21, YU12, YV12.
    /// Value says to video source class which pixel format preferable for
    /// output video frame. Particular video source class can ignore this params
    /// during initialization. Parameters should be set before initialization.
    std::string fourcc{"YUYV"};
    /// Frame width. User can set frame width before initialization
    /// or after. Some video source classes may set width automatically.
    int width{1920};
    /// Frame height. User can set frame height before
    /// initialization or after. Some video source classes may set height
    /// automatically.
    int height{1080};
    /// Gain mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    int gainMode{1};
    /// Gain value. Value: 0(min for particular video source class)
    /// - 65535(max for particular video source class).
    int gain{0};
```

```
    /// Exposure mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    int exposureMode{1};
    /// Exposure value. Value: 0(min for particular video source
    /// class) - 65535(max for particular video source class).
    int exposure{1};
    /// Focus mode. Focus mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    int focusMode{1};
    /// Focus position. Value: 0(full near) - 65535(full far).
    int focusPos{0};
    /// Video capture cycle time. **VSource** class sets this value
    /// automatically. This parameter means time interval between two captured
    /// video frame.
    int cycleTimeMks{0};
    /// FPS. User can set frame FPS before initialization or after.
    /// Some video source classes may set FPS automatically.
    float fps{0};
    /// Open flag. 0 - not open, 1 - open.
    bool isOpen{false};
    /// Custom parameter. Depends on implementation.
    float custom1{0.0f};
    /// Custom parameter. Depends on implementation.
    float custom2{0.0f};
    /// Custom parameter. Depends on implementation.
    float custom3{0.0f};

    JSON_READABLE(VSourceParams, logLevel, source, fourcc, width, height,
                  gainMode, exposureMode, focusMode, fps, custom1,
                  custom2, custom3);

    /**
     * @brief operator =
     * @param src Source object.
     * @return VSourceParams obect.
     */
    VSourceParams& operator= (const VSourceParams& src);
    /**
     * @brief Encode params. The method doesn't encode params:
     * source and fourcc fields.
     * @param data Pointer to data buffer.
     * @param size Size of data.
     */
    void encode(uint8_t* data, int& size);
    /**
     * @brief Decode params. The method doesn't decode params:
     * source and fourcc fields.
     * @param data Pointer to data.
     * @return TRUE is params decoded or FALSE if not.
     */
    bool decode(uint8_t* data);
};
```

Table 4 - VSourceParams class fields description.

Field	type	Description
logLevel	int	Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal, 3 - File and terminal.
source	string	Video source: file, video stream, video device, camera num, etc.
fourcc	string	FOURCC: RGB24, BGR24, YUYV, UYVY, GRAY, YUV24, NV12, NV21, YU12, YV12. Value says to video source class which pixel format preferable for output video frame. Particular video source class can ignore this params during initialization. Parameters should be set before initialization.
width	int	Frame width. User can set frame width before initialization or after. Some video source classes may set width automatically.
height	int	Frame height. User can set frame height before initialization or after. Some video source classes may set height automatically.
gainMode	int	Not supported.
gain	int	Gain value. Value: 0(min for particular video source class) - 65535(max for particular video source class).
exposureMode	int	Exposure mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.
exposure	int	Exposure value. Value: 0(min for particular video source class) - 65535(max for particular video source class).
focusMode	int	Focus mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.
focusPos	int	Focus position. Value: 0(full near) - 65535(full far).
cycleTimeMks	int	Video capture cycle time. **VSourceOpenCv** class sets this value automatically depending on input FPS value. This parameter means time interval between two captured video frame.
fps	float	FPS. User can set frame FPS before initialization or after. Some video source classes may set FPS automatically.
isOpen	bool	Open flag. false - not open, true - open.
custom1	float	Not supported.
custom2	float	Not supported.
custom3	float	Not supported..

None: *VSourceParams class fields listed in Table 4* ***must*** *reflect params set/get by methods setParam(...) and getParam(...).*

Example

Example shows how to use VSourceOpenCv. Example provides an application, which asks user for init method (0 - initString, 1 - config file), then for video source([video device or ID or file];[width];[height]) and if user want to display frames(y/n). After loading video source, example launches reading frame by frame. If user chose to display frames, images will appear as well as sliders to modify exposure time, gain or focus position. User can also see debug info about video source parameters. If display mode is off, application will print current frame number and also elapsed time between getting new frame.

```
#include <iostream>
#include "VSourceOpenCv.h"



// Link namespaces.
using namespace cr::utils;
using namespace cr::video;
using namespace std;
using namespace cv;



/// Trackbar focus position.
int g_trackbarFocusPosition = 0;
/// Trackbar gain.
int g_trackbarGain = 0;
/// Trackbar exposure.
int g_trackbarExposure = 0;




// Entry point.
int main(void)
{
    cout << "================================================\n";
    cout << "VSourceOpenCv v" << VSourceOpenCv::getVersion() << " Test\n";
    cout << "================================================\n";

    // Create video source object.
    VSource* videoSource = new VSourceOpenCv();

    // Set log level (optional). Print in console.
    videoSource->setParam(VSourceParam::LOG_LEVEL, 2);

    // Chose init method.
    int initMethod = 0;
    cout << "Chose init method (0 - initString, 1 - config file): ";
    cin >> initMethod;

    // Init video source by init string.
```

```
    if (initMethod == 0)
    {
        // Dialog to enter video source init string.
        string initString = "";
        cout << "Set init string (<source>;<width>;<height>): ";
        cin >> initString;

        // Open video source.
        if (!videoSource->openVSource(initString))
        {
            cout << "ERROR: Video source not open\n";
            return -1;
        }
    }
    // Init video source by config file.
    else
    {
        // Init params structure.
        VSourceParams params;

        // Read params from file.
        ConfigReader config;
        if(!config.readFromFile("VSourceOpenCvParams.json"))
        {
            cout << "Can't open config file. Default params created." << endl;
            config.set(params, "VSourceParams");
            config.writeToFile("VSourceOpenCvParams.json");
            return -1;
        }

        // Get params.
        if(!config.get(params, "VSourceParams"))
        {
            cout << "Invalid params format" << endl;
            return -1;
        }

        // Init video source.
        if (!videoSource->initVSource(params))
        {
            cout << "ERROR: Video source not open\n";
            return -1;
        }
    }

    // Dialog to enter video source init string.
    string displayMode = "";
    cout << "Do you want to display frames (y/n) ? : ";
    cin >> displayMode;

    // Init input frame object and params structure.
    Frame bgrFrame;
    VSourceParams params;

    // Init windows.
```

```
    namedWindow("VIDEO", WINDOW_AUTOSIZE);
    // Create focus trackbar.
    createTrackbar("Fix focus", "VIDEO", &g_trackbarFocusPosition, 65535);
    // Create gain trackbar.
    createTrackbar("Gain", "VIDEO", &g_trackbarGain, 65535);
    // Create gain exposure.
    createTrackbar("Exposure", "VIDEO", &g_trackbarExposure, 65535);

    // Main loop.
    int trackbarFocusPositionPrev = 0;
    int trackbarGainPrev = 0;
    int trackbarExposurePrev = 0;
    while (true)
    {
        // Wait 1 sec (1000 msec) for new frame.
        if (!videoSource->getFrame(bgrFrame, 1000))
        {
            cout << "WARNING: no frame\n";
            continue;
        }

        // Set proper modes to allow params modification.
        videoSource->setParam(VSourceParam::EXPOSURE_MODE, 0);
        videoSource->setParam(VSourceParam::FOCUS_MODE, 0);

        // Check trackbars values.
        if (trackbarFocusPositionPrev != g_trackbarFocusPosition)
        {
            videoSource->setParam(VSourceParam::FOCUS_POS, g_trackbarFocusPosition);
            trackbarFocusPositionPrev = g_trackbarFocusPosition;
        }
        if (trackbarGainPrev != g_trackbarGain)
        {
            videoSource->setParam(VSourceParam::GAIN, g_trackbarGain);
            trackbarGainPrev = g_trackbarGain;
        }
        if (trackbarExposurePrev != g_trackbarExposure)
        {
            videoSource->setParam(VSourceParam::EXPOSURE, g_trackbarExposure);
            trackbarExposurePrev = g_trackbarExposure;
        }

        // Get params.
        params = videoSource->getParams();

        // Display video.
        if (displayMode == "y" || displayMode == "Y")
        {
            // Prepare OpenCV image.
            Mat image(Size(bgrFrame.width, bgrFrame.height),
                      CV_8UC3, bgrFrame.data);

            // Draw params on video based on params structure.
            putText(image, "VSourceParams:",
            Point(5, 20), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255, 255, 0), 1);
```

```
putText(image, "Frame ID: " + to_string(bgrFrame.frameId),
Point(5, 40), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Elapse time,mks: "+to_string(params.cycleTimeMks),
Point(5, 60), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Width: " + to_string(params.width),
Point(5, 80), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Height: " + to_string(params.height),
Point(5, 100), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Gain mode: " + to_string(params.gainMode),
Point(5, 120), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Gain: " + to_string(params.gain),
Point(5, 140), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Focus mode (1): " + to_string(params.focusMode),
Point(5, 160), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Focus pos: " + to_string(params.focusPos),
Point(5, 180), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Exposure mode (2): "+to_string(params.exposureMode),
Point(5, 200), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Exposure: " + to_string(params.exposure),
Point(5, 220), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Fps: " + to_string(params.fps),
Point(5, 240), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);

// Draw params on video based on getParam() method.
putText(image, "getParam() method:",
Point(250, 20), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255, 255, 0), 1);
putText(image, "Frame ID: " + to_string(bgrFrame.frameId),
Point(250, 40), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Elapse time,mks: " +
to_string(videoSource->getParam(VSourceParam::CYCLE_TIME_MKS)),
Point(250, 60), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Width: " +
to_string(videoSource->getParam(VSourceParam::WIDTH)),
Point(250, 80), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Height: " +
to_string(videoSource->getParam(VSourceParam::HEIGHT)),
Point(250, 100), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Gain mode: " +
to_string(videoSource->getParam(VSourceParam::GAIN_MODE)),
Point(250, 120), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Gain: " +
to_string(videoSource->getParam(VSourceParam::GAIN)),
Point(250, 140), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Focus mode (1): " +
to_string(videoSource->getParam(VSourceParam::FOCUS_MODE)),
Point(250, 160), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Focus pos: " +
to_string(videoSource->getParam(VSourceParam::FOCUS_POS)),
Point(250, 180), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Exposure mode (2): " +
to_string(videoSource->getParam(VSourceParam::EXPOSURE_MODE)),
Point(250, 200), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
putText(image, "Exposure: " +
to_string(videoSource->getParam(VSourceParam::EXPOSURE)),
Point(250, 220), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);
```

```
            putText(image, "Fps: " +
            to_string(videoSource->getParam(VSourceParam::FPS)),
            Point(250, 240), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 255), 1);

            // Display video.
            imshow("VIDEO", image);

            // Process keyboard events.
            switch (waitKey(1))
            {
            // ESC - exit.
            case 27:
            {
                cout << "Exit application\n";
                return -1;
            }
            // 1 - Change focus mode.
            case 49:
            {
                // Change focus mode.
                if (params.focusMode < 0)
                    params.focusMode = 1;
                else
                    params.focusMode = 1 - params.focusMode;

                // Set focus mode.
                if (videoSource->setParam(VSourceParam::FOCUS_MODE,
                                          params.focusMode))
                {
                    cout << "Focus mode set" << endl;
                }
            }
            // 2 - Change exposure mode.
            case 50:
            {
                // Change focus mode.
                if (params.exposureMode < 0)
                    params.exposureMode = 1;
                else
                    params.exposureMode = 1 - params.exposureMode;

                // Set focus mode.
                if (videoSource->setParam(VSourceParam::EXPOSURE_MODE,
                                          params.exposureMode))
                {
                    cout << "Exposure mode set" << endl;
                }
            }
            }

            if (waitKey(1) == 27)
            {
                cout << "Exit application\n";
                return -1;
            }
```

```
        }
        else
        {
            // Display info.
            cout << "Frame ID: " << bgrFrame.frameId << ", elapsed time: " <<
            params.cycleTimeMks << " msec.\n";
        }
    }

    return 1;
}
```

Build and connect to your project

Typical commands to build **VSourceOpenCv** library:

```
git clone https://github.com/ConstantRobotics-Ltd/VSourceOpenCv.git
cd VSourceOpenCv
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want connect **VSourceOpenCv** library 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
```

You can add repository **VSourceOpenCv** as submodule by commands:

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

In you repository folder will be created folder **3rdparty/VSourceOpenCv** which contains files of **VSourceOpenCv** repository with subrepositories **VSource** and **Logger**. New structure of your repository:

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

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(${PARENT}_SUBMODULE_VSOURCE                         ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VSOURCE)
    SET(${PARENT}_VSOURCE_OPENCV                        ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VSOURCE_OPENCV_TEST                   OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

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

Done!