recorderopencv

VideoRecorderOpenCv C++ library

v1.1.1

Table of contents

Overview

VideoRecorderOpenCv C++ library provides methods for recording video to file's with controlling file's and folder size's. This library is built upon the OpenCV library. The library records **.avi** video files with "D.I.V.X" codec (build in OpenCV). The library is cross-platform and compatible with Windows and Linux OS. The library built with C++17 standard and doesn't have any third-party dependencies except OpenCV library.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	05.12.2022	- First version
1.1.0	13.12.2023	- Add async mode.
1.1.1	07.04.2024	- Documentation updated.

Library files

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

```
CMakeLists.txt ----------------------- Main CMake file of the library.
src ---------------------------------- Folder with library source code.
    CMakeLists.txt ------------------- CMake file.
    VideoRecorderOpenCv.h ------------ Main library header file.
    VideoRecorderOpenCv.cpp ---------- C++ implementation file.
    VideoRecorderOpenCvVersion.h ----- Header file with library version.
    VideoRecorderOpenCvVersion.h.in -- File for CMake to generate version header.
test --------------------------------- Folder for test application.
    CMakeLists.txt ------------------- CMake file to include test application.
    main.cpp ------------------------- Source code of test application.
```

VideoRecorderOpenCv class description

Class declaration

VideoRecorderOpenCv class declared in **VideoRecorderOpenCv.h** file. Class declaration:

```
class VideoRecorderOpenCv
{
public:

    /// Class constructor.
    VideoRecorderOpenCv();

    /// Class destructor.
    ~VideoRecorderOpenCv();

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

    /// Init video recorder.
    bool init(std::string initString);

    /// Write video frame.
    bool write(uint8_t* frameBgr, int width, int height);

    /// Stop video recording.
    void stopRecording();
};
```

getVersion method

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

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

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

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

Console output:

```
VideoRecorderOpenCv version: 1.1.1
```

init method

The **init(...)** method initializes video recorder. The method checks if destination folder exists. Method declaration:

```
bool init(std::string initString);
```

Parameter	Value
initString	Initialization string. Value: <folder>;<maxFolderSizeMB>;<maxFileSizeMB>;<prefix>;<fps>;<async>. Example: "video;2000;200;test;30;async". Initialization string parameters: **folder** - folder path to store video. The method will check if this folder exists and if not the method will create folder. **maxFolderSizeMB** - maximum folder size (size of all files) in megabytes. **maxFileSize** - Maximum video file size in megabytes. If the size of the currently recorded video file exceeds the specified size, the library will stop recording and create a new file. **prefix** - prefix for files names. Example: **prefix_10.12.11_10:30:20.avi**. **fps** - FPS of output file. This values will be recorder in ***.avi** video file header for playback. **async** (optional) - if this option exists in initialization string the library will run recording in separate thread. If not - the library will write in the same thread as the write(...) method.

Returns: TRUE if the video recorder is initialized or FALSE if not.

write method

The **write(...)** method writes video frame to output video file. In the case of asynchronous recording (**async** option in initialization string, init(...) method), the method will give the frame data to the recording thread and return control. In case of synchronous recording, the method will first perform recording and then return control. Method declaration:

```
bool write(uint8_t* frameBgr, int width, int height);
```

Parameter	Value
frameBgr	Pointer to video frame BGR data.
width	Video frame width.
height	Video frame height.

Returns: TRUE if the video frame was recorder or FALSE if not.

stopRecording method

The **stopRecording(...)** method stops video recording. The method stops write thread (in case asynchronous recording) and close output video file. Method declaration:

```
void stopRecording();
```

Example

Test application and show how to use **VideoRecorderOpenCv** object. Its open video file "test.mp4", asked user folder name, folder and file sizes and record file to folder according parameters:

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

using namespace cr::video;
using namespace std;
using namespace cv;

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

    // Init video video recorder.
    VideoRecorderOpenCv recorder;
    if (!recorder.init("Output;200;20;test;30;async"))
        return -1;
```

```
    // Main loop.
    Mat frameBgr;
    while (true)
    {
        // Capture video frame.
        videoSource >> frameBgr;
        if (frameBgr.empty())
        {
            videoSource.set(CAP_PROP_POS_FRAMES, 0);
            continue;
        }

        // Write frame.
        if (!recorder.write(frameBgr.data,
                            frameBgr.size().width,
                            frameBgr.size().height))
            cout << "Recording problem" << endl;

    }
    return 1;
}
```

Build and connect to your project

Typical commands to build **VideoRecorderOpenCv** library:

```
cd VideoRecorderOpenCv
mkdir build
cd build
cmake ..
make
```

If you want connect **VideoRecorderOpenCv** 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
```

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

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

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(${PARENT}_SUBMODULE_VIDEO_RECORDER_OPENCV           ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VIDEO_RECORDER_OPENCV)
    SET(${PARENT}_VIDEO_RECORDER_OPENCV                 ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VIDEO_RECORDER_OPENCV_TEST            OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

Done!