Motion Magnificator C++ library v2.0.0

programmer's manual

Table of contents

Overview

MotionMagnificator C++ library version **2.0.0** is a versatile library that allows users to enhance and 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). It does not rely on any third-party code or include additional software libraries. This library is suitable for various types of camera (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.

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.

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
README.md ----------------------------- Documentation
3rdparty ------------------------------ Folder with 3rdparty libraries
    CMakeLists.txt -------------------- CMake file for 3rdpary folder
    Frame ----------------------------- Files of Frame 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 app
    3rdaprty -------------------------- Folder with 3rdparty libraries
        CMakeLists.txt ---------------- CMake file for 3rdparty folder
        SimpleFileDialog -------------- File dialog service library
        VSourceOpenCv ----------------- Video capture service library
    main.cpp -------------------------- Source C++ file of demo app
example ------------------------------- Folder for simple example
    CMakeLists.txt -------------------- CMake file of example
    main.cpp -------------------------- Source C++ file of example
test ---------------------------------- Folder with test app (benchmark)
    CMakeLists.txt -------------------- CMake file of test app (benchmark)
    main.cpp -------------------------- Source C++ file of test app
```

MotionMagnificator library depends on open source **Frame** library (provides video frame structure and pixel formats description). Additionally demo application depends on open source **SimpleFileDialog** (provides dialog to open files) and **VSourceOpenCv** (provides method to capture video from files, cameras and streams, supplied as source code under **MotionMagnificator** license).

Key features and capabilities

Table 2 - Key features and capabilities.

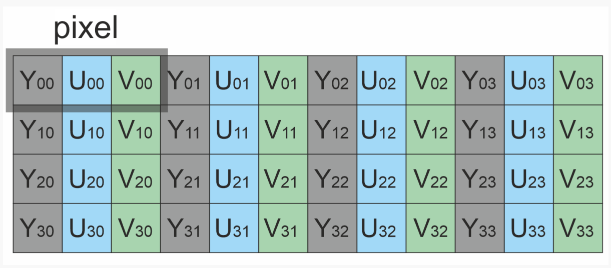
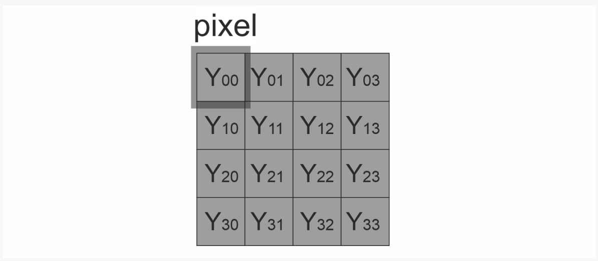
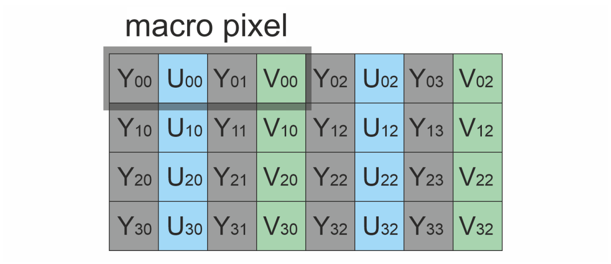
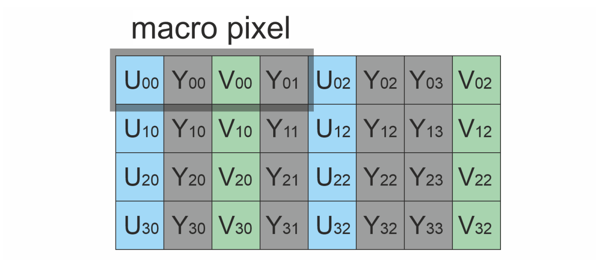
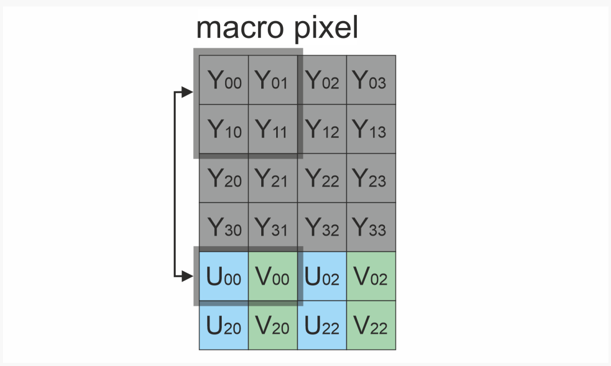
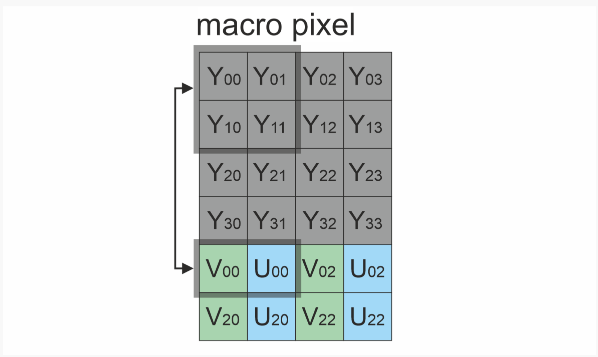
Parameter and feature	**Description**
Programming language	C++ (standard C++17). No third-party dependencies.
Supported OS	Compatible with any operating system that supports the C++ compiler (C++17 standard).
Movement type	The library is able to magnify any kind of movement, but it is best suitable for 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.
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.

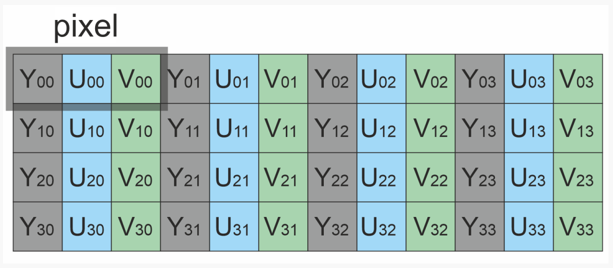
Supported pixel formats

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:

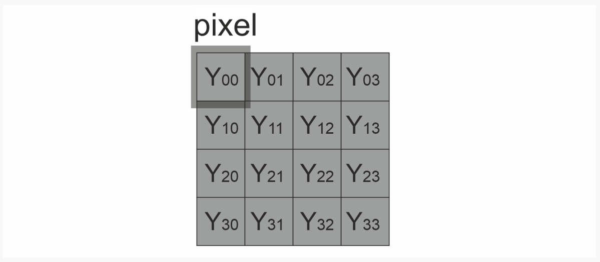
```
enum class Fourcc
{
    /// RGB 24bit pixel format.
    RGB24 = MAKE_FOURCC_CODE('R', 'G', 'B', '3'),
    /// BGR 24bit pixel format.
    BGR24 = MAKE_FOURCC_CODE('B', 'G', 'R', '3'),
    /// YUYV 16bits per pixel format.
    YUYV  = MAKE_FOURCC_CODE('Y', 'U', 'Y', 'V'),
    /// UYVY 16bits per pixel 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.

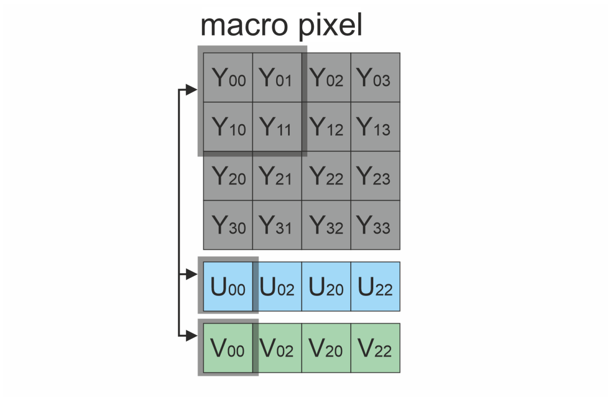
 **YUV24**	 **GRAY**
 **YUYV**	 **UYVY**
 **NV12**	 **NV21**


pixel
Y00 U00 V00 Y01 U01 V01 Y02 U02 V02 Y03 U03 V03
Y10 U10 V10 Y11 U11 V11 Y12 U12 V12 Y13 U13 V13
Y20 U20 V20 Y21 U21 V21 Y22 U22 V22 Y23 U23 V23
Y30 U30 V30 Y31 U31 V31 Y32 U32 V32 Y33 U33 V33

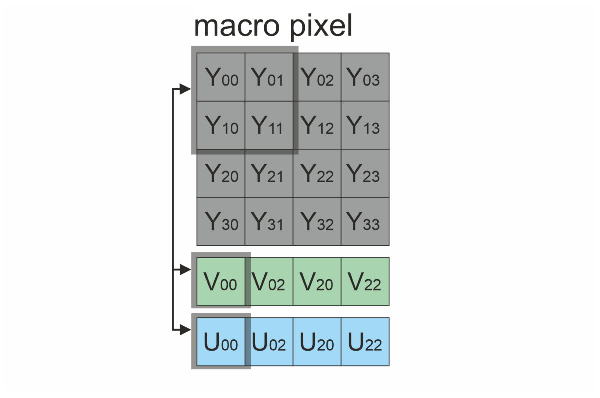
YUV24



GRAY



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. Creating the integral image.
3. Calculating pixel intensity values' differences.
4. Filtrating signal and applying amplification.
5. 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:

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

    /// Class constructor.
    MotionMagnificator();

    /// Class destructor.
    ~MotionMagnificator();

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

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

    /// Execute command.
    bool executeCommand(MotionMagnificatorCommand id);

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

    /// Set magnification mask.
    bool setMask(cr::video::Frame mask);
}
```

getVersion method

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: 2.0.0
```

setParam method

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(MotionMagnificatorParam id, float value);
```

Parameter	Description
id	Parameter ID according to **MotionMagnificatorParam** enum.
value	Parameter value. Value depends on parameter ID.

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

getParam method

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(MotionMagnificatorParam id);
```

Parameter	Description
id	Parameter ID according to **MotionMagnificatorParam** enum.

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

executeCommand method

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(MotionMagnificatorCommand id);
```

Parameter	Description
id	Command ID according to **MotionMagnificatorCommand** enum.

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

processFrame method

processFrame(...) method designed to perform magnification algorithm. Method declaration:

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

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 (see **MotionMagnificatorParam** enum description) the method should return TRUE.

setMask method

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);
```

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).

Data structures

MotionMagnificatorCommand enum

Enum declaration:

```
enum class MotionMagnificatorCommand
{
    RESET = 1,
    ON = 2,
    OFF = 3
};
```

Table 4 - Motion magnification commands description.

Command	Description
RESET	Reset algorithm. Clears previous filter values.
ON	Enable motion magnificator. Input Frame will be processed.
OFF	Disable motion magnificator. If the magnificator is not activated, frame processing is not performed - frame is only forwarded.

MotionMagnificatorParam enum

Enum declaration:

```
enum class MotionMagnificatorParam
{
    /// Mode. Default: 0 - Off, 1 - On.
    MODE = 1,
    /// Amplification factor adjusts the intensity of motion enhancement in
    /// videos, with higher values emphasizing motion changes
    /// and lower values maintaining the original motion characteristics.
    AMPLIFICATION = 2
};
```

Table 5 - MotionMagnificator class 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.
AMPLIFICATION	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.

Build and connect to your project

Typical commands to build **MotionMagnificator** library:

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

If you want to connect **MotionMagnificator** library to your CMake project as source code, you can do the following. For example, if your repository has structure:

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

You can add repository **MotionMagnificator** as git submodule by commands (only if you have access to GitHub repository):

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

In your repository folder, a new **3rdparty/MotionMagnificator** folder will be created, which contains files from **MotionMagnificator** repository along with its subrepository **Frame**. If you don't have access to GitHub repository, copy **MotionMagnificator** repository folder to **3rdparty** folder to your repository. The new structure of your repository will be as follows:

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

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

```
cmake_minimum_required(VERSION 3.13)

################################################################################
## 3RD-PARTY
## dependencies for the project
################################################################################
project(3rdparty LANGUAGES CXX)

```

```
###############################################################################
## SETTINGS
## basic 3rd-party settings before use
###############################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

###############################################################################
## CONFIGURATION
## 3rd-party submodules configuration
###############################################################################
SET(${PARENT}_SUBMODULE_MOTION_MAGNIFICATOR             ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_MOTION_MAGNIFICATOR)
    SET(${PARENT}_MOTION_MAGNIFICATOR                   ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_MOTION_MAGNIFICATOR_TEST              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" and copies the video frame data into an object of the Frame class and performs motion magnification.

```
#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;

    // Create frames.
    cv::Mat frameBgrOpenCv;
    cv::Mat frameYUVOpenCv;
    cr::video::Frame frameYUV =
    cr::video::Frame(int(videoSource.get(cv::CAP_PROP_FRAME_WIDTH)),
        int(videoSource.get(cv::CAP_PROP_FRAME_HEIGHT)),
                     cr::video::Fourcc::YUV24);
    // Create object.
    cr::mmag::MotionMagnificator magnificator;

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

        cv::cvtColor(frameBgrOpenCv, frameYUVOpenCv, cv::COLOR_BGR2YUV);
        // Copy frame data from OpenCv frame to Frame.
        std::memcpy(frameYUV.data, frameYUVOpenCv.data, frameYUV.size);

        // Magnify video.
        magnificator.processFrame(frameYUV);

        // Copy back and convert back.
        std::memcpy(frameYUVOpenCv.data, frameYUV.data, frameYUV.size);
        cv::cvtColor(frameYUVOpenCv, frameBgrOpenCv, cv::COLOR_YUV2BGR);

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

        // Wait ESC.
        if (cv::waitKey(1) == 27)
```

```
            return -1;
    }

    return 1;
}
```