VOverlay interface C++ class

v1.1.0

Overview

VOverlay C++ library provides standard interface for overlaying information on video for different implementations. Standard for overlaying information on video used in video processing pipeline interface. Video processing pipeline interface understands only interfaces and user can make custom video overlay class implementation depends on situation. The **VOverlay** interface provides only one method **overlay(...)** to overlay information on video. This method is called by pipelines for each frame of the video. Any implementation may include additional methods depending on the situation. The method depends on the **Frame** class, which defines the data structure of the video frame.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	31.08.2023	First version.
1.1.0	13.12.2023	Virtual destructor added. Frame class updated.

VOverlay class description

VOverlay class declaration

VOverlay.h file contains **VOverlay** class declaration. Class declaration:

```
class VOverlay
{
public:
    /**
     * @brief Class destructor.
     */
    virtual ~VOverlay();
    /**
```

```
     * @brief Get string of current class version.
     * @return String of current class version "Major.Minor.Patch"
     */
    static std::string getVersion();
    /**
     * @brief Overlay the information on the video.
     * @param frame Frame to put information.
     * @param data Pointer to custom data. Depends on implementation.
     * @return TRUE if everything OK or FALSE in case any errors.
     */
    virtual bool overlay(cr::video::Frame& frame, void* data = nullptr) = 0;
};
```

getVersion method

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

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

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

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

Console output:

```
VOverlay class version: 1.1.0
```

overlay method

overlay(...) method overlays custom information on video. Method declaration:

```
virtual bool overlay(cr::video::Frame& frame, void* data = nullptr) = 0;
```

Parameter	Description
frame	Video frame object to overlay information. Each video overlay implementation should support all RAW pixel format declared in **Frame** class (RGB24, BGR24, YUYV, UYVY, GRAY, YUV24, NV12, NV21, YU12, YV12).
data	Pointer to information structure to overlay. User defines data structure format depends on implementation.

Returns: TRUE if information overlayed or FALSE if not (not supported frame format, invalid frame data etc.).

Build and connect to your project

Typical commands to build **VOverlay** library:

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

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

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

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

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

In your repository folder **3rdparty/VOverlay** will be created which contains files of **VOverlay** repository with subrepository **Frame**. New structure of your repository:

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

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_VOVERLAY                        ON  CACHE BOOL "" FORCE)

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

File **3rdparty/CMakeLists.txt** adds folder **VOverlay** to your project. Your repository new structure will be:

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

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

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

Done!