RMT® MACHINE VISION

RF_MOTION_DETECTOR

SOFTWARE LIBRARY FOR AUTOMATIC DETECTION OF MOVING OBJECTS ON VIDEO
(Programmer manual)

Software library version: 1.0
Software library release date: 23.10.2017
Document version: 1.0
Document release date: 23.10.2017

www.rmtvision.com

CONTENTS

DOCUMENT VERSIONS

Table 1 – Document versions.

Version	Description
1.0	Description of the RF_MOTION_DETECTOR software library, version 1.0.

SOFTWARE LIBRARY VERSIONS

Table 2 – Software library versions.

Version	Description
1.0	The first version of the RF_MOTION_DETECTOR software library.

DESCRIPTION

The RF_MOTION_DETECTOR software library implements the algorithm for automatic detection of moving objects on video. The software library works with images obtained by cameras of any spectral range. The algorithm implemented in the software library is equally effective for detecting both air objects and land ones. The software library allows you to set parameters for detecting the objects in accordance with the requirements of the observed situation. The source code of the software library is written in **C++**. Its **C** language version is available on request. The software library is supplied as source code files containing the description of the **RfMotionDetector** software class of the detection algorithm. One instance of the software class allows implementation of one video processing channel. To build multi-channel systems (systems with multiple video sources), you need to create several instances of the **RfMotionDetector** class. The software library includes the following files: RfMotionDetector.h (declaration of the RfMotionDetector class and data structures), RfMotionDetector.cpp, RfForegroundDetector.h (declaration of helper classes), RfForegroundDetector.cpp, RfBoundingBoxFinder.h (declaration of helper classes), RfBoundingBoxFinder.cpp, RfTrackDetector.h (declaration of helper classes), RfTrackDetector.cpp, Rects.h (declaration of data structures). To use the RF_MOTION_DETECTOR software library in the project, the developer should include the above source code files in the project.

BASIC FEATURES AND CAPABILITIES

Table 3 summarizes the main characteristics of the software library and detection algorithm.

Table 3 – Main features of the software library and detection algorithm.

Parameter	Value and Notes
Software library language	C++ (C++98 standard) without the use of third-party software libraries and functions dependent on the operating system. The C language version (C99 standard) is available on request.
Compatibility with operating systems	Compatible with all 32 and 64bit operating systems for C++ projects.
Compatibility with compilers	Compatible with any compilers that support the C++98 standard.
Amount of statically allocated memory	No more than 32 Kbyte of statically allocated memory.
Amount of dynamically allocated memory	The amount of dynamically allocated memory is determined during the software library initialization, and depends on the size of the processed video frames. The approximate amount of dynamically allocated memory for the video of the required resolution can be estimated using the demonstration program.
Number of detector channels	The software class implements one detector channel (processing of one video source).
Maximum dimensions of detected objects	Limited by the frame size.
Minimum dimensions of detected objects	8x8 pixels.

Parameter	Value and Notes
Discreteness of computation of the object coordinates	No more than 1 pixel. The algorithm calculates the detection rectangle for each detected object.
Format of the input video data	The library accepts video frames in the **mono_8** format (1 byte per pixel in grayscale).
Input video frame dimensions	Limited by RAM.
Computation speed	Calculations are made for each video frame independently of the previous frame after the relevant method of the RfMotionDetector software class is called. The software library does not contain any time functions and does not control the period of calling of processing functions. No requirements to the duration (time interval between frames). The processing time of one video frame does not depend on its size (it must lie within the minimum and maximum limits), it rather depends on the specific features and parameters of the C ++ (C) project which makes use of the library. The performance also depends on the computing platform where calculations are performed. To evaluate the library performance on the user's platform, the demonstration program is supplied. To estimate the calculation speed, test applications can be provided on request.
Shape and configuration of detected objects	The algorithm detects the moving objects of any type and shape. The detection algorithm implemented in the library does not recognize and identify objects, it only detects them.
Type of the implemented detection algorithm	Implemented the algorithm for estimating the background, based on the creation of multiple hypotheses along with the analysis of trajectories of the detected objects.

NOTE. All the values and parameters given in this document are relevant only for the image with the window coordinate system (the beginning in the upper left corner).

Quality of the automatic detection of moving objects depends on the conditions of observation and parameters of the object (its shape and contrast relative to background, and others). To assess the quality of detection in a variety of situations, the demonstration program is provided.

OPERATION PRINCIPLE OF THE SOFTWARE LIBRARY

Operation principle of the detection algorithm

The software library processes each video frame. To process the next video frame, the developer must pass a pointer to the frame data into the processing function. Video frames are processed in two stages: detection of groups of pixels that are not related to the background, and analysis of object trajectories. The operation principle of the algorithm for detecting groups of pixels of the objects is based on the creation of hypotheses about the background situation on video frames and on the selection of objects (groups of pixels), which can not be referred to the background due to their statistical characteristics. The operation principle of the algorithm for analysis of trajectories is based on the comparison of the detected objects from frame to frame, and on the analysis of motion parameters.

Principles of use of the software library

To use the software library the developer should include the library files (RfMotionDetector.h, RfMotionDetector.cpp, RfForegroundDetector.h, RfForegroundDetector.cpp, RfBoundingBoxFinder.h, RfBoundingBoxFinder.cpp, RfTrackDetector.h, RfTrackDetector.cpp, Rects.h) into the project to be designed. Then, the developer should create an object of the **RfMotionDetector** class, declared in the **RfMotionDetector.h** file. A typical sequence of use of the software library can be as follows:

1. Create an object of the RfMotionDetector class, and pass a structure of parameters of the detection algorithm to the class constructor as the parameter (if a constructor is not used by default).
2. If the constructor was used by default when creating the object, set parameters of the detection algorithm by calling the *Init(…)* method.
3. To process the next frame and to obtain results, call the *Execut(…)* method.

4. To reset the results of processing, call the *Reset()* method.

DETECTION ALGORITHM CONSTANTS

The algorithm constants are declared in the **RfMotionDetector.h** file and can be changed by the developer if necessary. These constants define the values of detection algorithm parameters set by default. Constants of the detection algorithm are listed in Table 5.

Table 5 – Constants of the detection algorithm.

Constant name and description	Default value
RF_DEFAULT_MIN_OBJECT_W – the minimum width of the detected object (rectangle around the object) by default.	2 pixels
RF_DEFAULT_MIN_OBJECT_H – the minimum height of the detected object (rectangle around the object) by default.	2 pixels
RF_DEFAULT_MAX_OBJECT_W – the maximum width of the detected object (rectangle around the object) by default.	256 pixels
RF_DEFAULT_MAX_OBJECT_H – the maximum height of the detected object (rectangle around the object) by default.	256 pixels
RF_DEFAULT_MAX_OBJECT_VELOCITY – the maximum velocity of the object (pixel/frame) in any direction (moving in one frame) at which some object detected on the previous frame will be identified with the object on the current frame by default.	15 pixels
RF_DEFAULT_TRACK_APPEARANCE_CRITERION – the object detection criterion by default. This criterion has the following meaning: in order for the object to be detected, it must be identified continuously (each frame without omissions) in the frame sequence that has the length not less than this value.	6 pixels
RF_DEFAULT_TRACK_DIRECTION_CRITERION – the object detection criterion by default, based on the direction of motion. This criterion has the following meaning: in order for the object to be detected, it must move in one direction continuously at least the specified number of frames (without omissions) at any speed below the maximum allowed.	15 frames

DATA STRUCTURES USED IN THE SOFTWARE LIBRARY

Review of data structures

The software library returns information about the detected moving objects in the form of rectangles. The data structures for rectangles are declared in the **Rects.h** file. The data structures, which define parameters of the detection algorithm, are declared in the **RfMotionDetector.h** file. Below is the declaration of data structures contained in the **Rects.h** file:

```
typedef struct {
    int x;
    int y;
} Point;
```

```
typedef struct {
    Point topLeft;
    Point bottomRight;
} Rect;
```

Table 6 describes the fields of the structures Point and Rect.

Table 6 – Description of the fields of the structures Point and Rect.

Field name	Description
x	Horizontal coordinate.
y	Vertical coordinate.
topLeft	Coordinates of the upper left corner of the object rectangle.
bottomRight	Coordinates of the lower right corner of the object rectangle.

Below is the declaration of the RfMotionDetectorParams structure contained in the **RfMotionDetector.h** file:

```
typedef struct {
    int frameW;
    int frameH;
    int minObjectW;
    int minObjectH;
    int maxObjectW;
    int maxObjectH;
    int maxObjectVelocity;
    int trackAppearanceCriterion;
    int trackDirectionCriterion;
} RfMotionDetectoParams;
```

Table 7 describes the fields of the RfMotionDetectorParams structure:

Table 7 – Description of the RfMotionDetectorParams structure fields.

Field name	Description
frameW	Width of the processed video frames.
frameH	Height of the processed video frames.
minObjectW	Minimum width of the object (rectangle) to be detected. Objects of a smaller width in pixels will be discarded.
minObjectH	Minimum height of the object (rectangle) to be detected. Objects of a smaller height in pixels will be discarded.
maxObjectW	Maximum width of the object (rectangle) to be detected. Objects of a greater width in pixels will be discarded.
maxObjectH	Maximum height of the object (rectangle) to be detected. Objects of a greater height in pixels will be discarded.
maxObjectVelocity	Maximum velocity of the object (pixel/frame) in any direction (moving in one frame) at which some object detected on the previous frame will be identified with the object on the current frame.
trackAppearanceCriterion	The object detection criterion. This criterion has the following meaning: in order for the object to be detected, it must be identified continuously (each frame without omissions) in the frame sequence that has the length not less than this value.
trackDirectionCriterion	The object detection criterion based on the direction of motion. This criterion has the following meaning: in order for the object to be detected, it must move in one direction continuously at least the specified number of frames (without omissions) at any speed below the maximum allowed.

DESCRIPTION OF SOFTWARE LIBRARY INTERFACE METHODS

List of the RfMotionDetector class methods

The RfMotionDetector software class interface is a set of methods available to the user. Table 8 lists the interface methods for the class.

Table 8 – List of methods of the RfMotionDetector class.

Method name	Description
RfMotionDetector()	Class constructor.
RfMotionDetector(…)	Class constructor with the transmission of algorithm parameters.
~RfMotionDetector()	Class destructor.
Init(…);	Method of software library initialization.
Execut(…)	Method of processing the next frame.
GetMaskImage(…)	Method of obtaining the motion mask (binary image where bright pixels indicate motion in the frame).
Reset()	Method of resetting the detection results.

RfMotionDetector(…) class constructor

The RfMotionDetector class constructor has two implementations: with the transmission of algorithm parameters and without transmission. Below is the declaration of the class constructor with the transmission of algorithm parameters.

Declaration:

```
RfMotionDetector(RfMotionDetectoParams params);
```

Parameters:

params	Structure of the detection algorithm parameters.

Init(…) method

The **Init(…)** method is intended for setting parameters of the detection algorithm. This method must be called before the others, if the algorithm parameters have not been set using the class constructor.

Declaration:

```
void Init(RfMotionDetectoParams params);
```

Parameters:

params	Structure of the detection algorithm parameters.

Execut(…) method

The **Execut(…)** method is intended for processing the next video frame. This method must be called to process each video frame. **Note:** only frames with the same parameters (video source, format and resolution) are allowed to be transmitted for processing.

Declaration:

```
std::vector<Rect> Execut(unsigned char *frame, unsigned char *frameMask = 0);
```

Parameters:

*frame	The pointer to the frame data. The frame size should be equal to (frameW * frameH). The frame pixels should be in the **mono_8** format (one byte per pixel in grayscale).
*frameMask	The frame mask. If necessary, a developer can set the ROI area (the area where the analysis should be performed). To do this, the developer must transmit the frame mask in the **mono_8** format (one byte per pixel - a binary image). The size of the transmitted mask should be equal to the size of the frame. Pixels having the value of 255 in the frame mask indicate that the corresponding pixels of the motion mask have not been taken into account in the calculation.

Return value:
The method returns a vector of detected objects (rectangles). If nothing is detected, the method returns a zero-length vector.

GetMaskImage(…) method

The **GetMaskImage**(…) method is intended to obtain a binary image of the motion mask that contains primary information about motion in the frame for the algorithm of processing the trajectory information. If a developer wants to implement his own algorithm for searching for objects using the motion mask, the library allows to get the motion mask. After control returns to the calling thread, the buffer transmitted by the pointer will be filled with the motion mask data.

Declaration:

```
void GetMaskImage(unsigned char *mask);
```

Parameters:

*mask	A pointer to the mask data array to be filled by the method. The mask pixels should be in the *mono_8* format (one byte per pixel). The size of the mask buffer should be equal to the size of the processed video frames.

Reset() method

The **Reset()** method is intended to reset the detection results. This method performs zeroing of statistical characteristics of the motion mask of trajectory information results. After calling this method, the detection algorithm starts to process video frames the same way as when processing the first video frame.

Declaration:

```
void Reset();
```

EXAMPLE OF USE OF THE SOFTWARE LIBRARY

The example below represents a source code of a console program that opens a video file, sends it to display, sends the motion mask to display, and displays the detection results. The example was designed for Windows operating systems. If you want to use the example for Linux operating systems, delete the operating system-dependent functions. An OpenCV computer vision library is used to capture and display video frames and to perform drawing tasks.

Listing:

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
#include <opencv2/opencv.hpp>
#include <Windows.h>
#include <shobjidl.h>
#include <time.h>
#include "RfMotionDetector.h"

// Video capture structure
typedef struct {
    cv::VideoCapture cap;
    cv::Mat frame;
    cv::Mat mono_frame;
    cv::Mat buf_frame;
    int frame_w;
    int frame_h;
    int fps;
} VIDEOCAPTURE;
```

```
// Global vars
VIDEOCAPTURE cap;

// Function Prototypes
bool OpenVideoFileDialog();

// Entry point
int main(int argc, char **argv) {

    std::cout << "RF_MOTION_DETECTOR v1.0 DEMO APP" << std::endl;

    // Open video file
    if (!OpenVideoFileDialog()) {
        std::cout << "ERROR: File not opened" << std::endl;
        Sleep(1000);
        return 0;
    }//if...

    std::cout << std::endl;
    std::cout << "CONTROL BUTTONS:" << std::endl;
    std::cout << "U - update motion mask" << std::endl;
    std::cout << "R - start/stop record" << std::endl << std::endl;
    std::cout << "TO CHANGE DETECTOR PARAMS Please edit DetectorParams.txt file !!!!!!!!!!!" << std::endl;

    // Create windows
    cv::namedWindow("RF_MOTION_DETECTOR v1.0 DEMO APP", CV_WINDOW_AUTOSIZE);
    cv::moveWindow("RF_MOTION_DETECTOR v1.0 DEMO APP", 20, 20);
    cv::namedWindow("MOTION MASK", CV_WINDOW_AUTOSIZE);
    cv::moveWindow("MOTION MASK", cap.frame_w + 40, 20);

    // Vars for video record
    int video_clip_num = 0;
    cv::VideoWriter *writer = 0;

    int video_start_time;
    int video_delta_time;

    // Init detector
    rf::RfMotionDetector detector;
    std::vector<rf::Rect> objects;
    rf::RfMotionDetectoParams detectorParams;
    detectorParams.frameW = cap.frame_w;
    detectorParams.frameH = cap.frame_h;

    // Load params from file
    std::string params_file_name(argv[0]);
    unsigned int pos_1 = static_cast<unsigned int>(params_file_name.rfind('\\'));
    params_file_name = params_file_name.substr(0, pos_1 + 1) + "DetectorParams.txt";
    std::ifstream F;
    F.open(params_file_name.c_str(), std::ios::in);
    if (F) {

        std::cout << "Params file loaded " << params_file_name << std::endl;

        std::ostringstream out;
        out << F.rdbuf();
        std::string text = out.str();

        std::string str("MINIMUN OBJECT WIDTH:");
        unsigned int pos_1 = static_cast<unsigned int>(text.find(str));
        unsigned int pos_2 = static_cast<unsigned int>(text.find(";", pos_1 + str.size()));
        if (pos_1 != text.npos && pos_2 != text.npos) {
            try {
                detectorParams.minObjectW = std::stoi(text.substr(pos_1 + str.size(), pos_2 - pos_1 - str.size()));
            }
            catch (...) {
                detectorParams.minObjectW = RF_DEFAULT_MIN_OBJECT_W;
            }//try...
        }//...
```

```
str = "MINIMUN OBJECT HEIGHT:";
pos_1 = static_cast<unsigned int>(text.find(str));
pos_2 = static_cast<unsigned int>(text.find(";", pos_1 + str.size()));
if (pos_1 != text.npos && pos_2 != text.npos) {
    try {
        detectorParams.minObjectH = std::stoi(text.substr(pos_1 + str.size(), pos_2 - pos_1 - str.size()));
    }
    catch (...) {
        detectorParams.minObjectH = RF_DEFAULT_MIN_OBJECT_H;
    }//try...
}//...

str = "MAXIMUM OBJECT WIDTH:";
pos_1 = static_cast<unsigned int>(text.find(str));
pos_2 = static_cast<unsigned int>(text.find(";", pos_1 + str.size()));
if (pos_1 != text.npos && pos_2 != text.npos) {
    try {
        detectorParams.maxObjectW = std::stoi(text.substr(pos_1 + str.size(), pos_2 - pos_1 - str.size()));
    }
    catch (...) {
        detectorParams.maxObjectW = RF_DEFAULT_MAX_OBJECT_W;
    }//try...
}//...

str = "MAXIMUM OBJECT HEIGHT:";
pos_1 = static_cast<unsigned int>(text.find(str));
pos_2 = static_cast<unsigned int>(text.find(";", pos_1 + str.size()));
if (pos_1 != text.npos && pos_2 != text.npos) {
    try {
        detectorParams.maxObjectH = std::stoi(text.substr(pos_1 + str.size(), pos_2 - pos_1 - str.size()));
    }
    catch (...) {
        detectorParams.maxObjectH = RF_DEFAULT_MAX_OBJECT_H;
    }//try...
}//...

str = "MAXIMUM OBJECT VELOCITY:";
pos_1 = static_cast<unsigned int>(text.find(str));
pos_2 = static_cast<unsigned int>(text.find(";", pos_1 + str.size()));
if (pos_1 != text.npos && pos_2 != text.npos) {
    try {
        detectorParams.maxObjectVelocity = std::stoi(text.substr(pos_1 + str.size(), pos_2 - pos_1 - str.size()));
    }
    catch (...) {
        detectorParams.maxObjectVelocity = RF_DEFAULT_MAX_OBJECT_VELOCITY;
    }//try...
}//...

str = "TRACK APPEARANCE CRITERION:";
pos_1 = static_cast<unsigned int>(text.find(str));
pos_2 = static_cast<unsigned int>(text.find(";", pos_1 + str.size()));
if (pos_1 != text.npos && pos_2 != text.npos) {
    try {
        detectorParams.trackAppearanceCriterion = std::stoi(text.substr(pos_1 + str.size(), pos_2 - pos_1 - str.size()));
    }
    catch (...) {
        detectorParams.trackAppearanceCriterion = RF_DEFAULT_TRACK_APPEARANCE_CRITERION;
    }//try...
}//...

str = "TRACK DIRECTION CRITERION:";
pos_1 = static_cast<unsigned int>(text.find(str));
pos_2 = static_cast<unsigned int>(text.find(";", pos_1 + str.size()));
if (pos_1 != text.npos && pos_2 != text.npos) {
    try {
        detectorParams.trackDirectionCriterion = std::stoi(text.substr(pos_1 + str.size(), pos_2 - pos_1 - str.size()));
    }
    catch (...) {
```

```
                detectorParams.trackDirectionCriterion = RF_DEFAULT_TRACK_DIRECTION_CRITERION;
            }//try...
        }//...

    }
    else {

        std::cout << "Params File not loaded " << params_file_name << std::endl;

        detectorParams.minObjectW = RF_DEFAULT_MIN_OBJECT_W;
        detectorParams.minObjectH = RF_DEFAULT_MIN_OBJECT_H;
        detectorParams.maxObjectW = RF_DEFAULT_MAX_OBJECT_W;
        detectorParams.maxObjectH = RF_DEFAULT_MAX_OBJECT_H;
        detectorParams.maxObjectVelocity = RF_DEFAULT_MAX_OBJECT_VELOCITY;
        detectorParams.trackAppearanceCriterion = RF_DEFAULT_TRACK_APPEARANCE_CRITERION;
        detectorParams.trackDirectionCriterion = RF_DEFAULT_TRACK_DIRECTION_CRITERION;

    }//if...

    // Init detector params
    detector.Init(detectorParams);

    cv::Mat maskImage = cv::Mat(cv::Size(cap.frame_w, cap.frame_h), CV_8U);

    while (true) {

        // Get time point
        video_start_time = static_cast<int>(clock());

        // Video frame capture
        cap.cap >> cap.frame;
        if (cap.frame.empty()) {
            cap.cap.set(CV_CAP_PROP_POS_FRAMES, 0);
            if (writer != 0) {
                writer->release();
                writer = 0;
            }//if...
            continue;
        }//if...

        // Convert to grayscale
        if (cap.frame.channels() == 1) memcpy(&cap.mono_frame.data[0], &cap.frame.data[0], cap.frame_w * cap.frame_h);
        else cv::cvtColor(cap.frame, cap.mono_frame, CV_RGB2GRAY);

        unsigned int startTime = static_cast<unsigned int>(clock());
        objects = detector.Execut((unsigned char *)cap.mono_frame.data);

        unsigned int processingTime = static_cast<unsigned int>(clock());
        std::string str = "Time = " + std::to_string(processingTime - startTime) + " ms";
        cv::putText(cap.frame, str, cv::Point(cap.frame_w - 140, 20), cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(255,
255, 0));

        detector.GetMaskImage(maskImage.data);

        // Draw info
        for (int i = 0; i < objects.size(); ++i) {
            cv::Rect2i rect(objects[i].topLeft.x - 5, objects[i].topLeft.y - 5, abs(objects[i].bottomRight.x - objects[i].topLeft.x +
10), abs(objects[i].bottomRight.y - objects[i].topLeft.y + 10));
            cv::rectangle(cap.frame, rect, cv::Scalar(0, 0, 255), 2);
        }//for...

        // Recor video
        if (writer != 0) {
            writer->write(cap.frame);
            cv::circle(cap.frame, cv::Point(20, 20), 10, cv::Scalar(0, 0, 255), CV_FILLED);
        }//if...

        // Show image
        cv::imshow("RF_MOTION_DETECTOR v1.0 DEMO APP", cap.frame);
```

```
        cv::imshow("MOTION MASK", maskImage);

        // Keyboard events
        switch (cv::waitKey(1)) {
        case 27:
            if (writer != 0) {
                writer->release();
                writer = 0;
            }//if...
            cv::destroyAllWindows();
            return 0;
        // U - reset
        case 85: detector.Reset(); break;
        // u - reset
        case 117: detector.Reset(); break;
        // R - start/stop record
        case 114:
            if (writer != 0) {
                writer->release();
                writer = 0;
            }
            else {
                std::string videoFileName = "record_" + std::to_string(video_clip_num++) + ".avi";
                writer = new cv::VideoWriter(videoFileName, -1, 25, cv::Size(cap.frame_w, cap.frame_h), true);
                assert(writer != 0);
            }//if...
            break;
        }//switch...

        // Wait
        video_delta_time = static_cast<int>(clock()) - video_start_time;
        int wait_time = 33 - video_delta_time;
        if (wait_time < 0) wait_time = 1;
        Sleep(wait_time);

    }//while...

    return 1;

}//int main...


bool OpenVideoFileDialog() {

    IFileOpenDialog *pFileOpen;
    HRESULT hr;
    PWSTR file;
    char* filename;

    hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED | COINIT_DISABLE_OLE1DDE);
    if (SUCCEEDED(hr)) {
        hr = CoCreateInstance(CLSID_FileOpenDialog,
            NULL, CLSCTX_ALL,
            IID_IFileOpenDialog,
            reinterpret_cast<void**>(&pFileOpen));
        if (SUCCEEDED(hr)) {
            pFileOpen->SetTitle(L"OPEN VIDEO FILE");
            hr = pFileOpen->Show(NULL);
            if (SUCCEEDED(hr)) {
                IShellItem *pItem;
                hr = pFileOpen->GetResult(&pItem);
                if (SUCCEEDED(hr)) {
                    hr = pItem->GetDisplayName(SIGDN_FILESYSPATH, &file);
                    if (SUCCEEDED(hr)) {
                        pItem->Release();
                        int count = WideCharToMultiByte(CP_ACP, 0, file, static_cast<int>(wcslen(file)), 0, 0, NULL, NULL);
                        filename = new char[count + 1];
                        WideCharToMultiByte(CP_ACP, 0, file, count, filename, count + 1, NULL, NULL);
                        filename[static_cast<size_t>(count)] = '\0';
```

```
                        std::cout << filename << std::endl;
                        cap.cap.open(filename);
                        if (!cap.cap.isOpened()) {
                            return false;
                        }
                        else {
                            cap.frame_w = static_cast<int>(cap.cap.get(CV_CAP_PROP_FRAME_WIDTH));
                            cap.frame_h = static_cast<int>(cap.cap.get(CV_CAP_PROP_FRAME_HEIGHT));
                            cap.fps = static_cast<int>(cap.cap.get(CV_CAP_PROP_FPS));
                            cap.cap >> cap.frame;
                            int num_frames = static_cast<int>(cap.cap.get(CV_CAP_PROP_FRAME_COUNT));
                            cap.mono_frame = cv::Mat(cv::Size(cap.frame_w, cap.frame_h), CV_8U);
                            cap.buf_frame = cv::Mat(cv::Size(cap.frame_w, cap.frame_h), cap.frame.channels() == 1 ? CV_8U
: CV_8UC3);
                            return true;
                        }//if...
                    }//if...
                }//if...
                pItem->Release();
            }//if...
        }//if...
    }//if...
    return false;

}//bool OpenVideoFileDialog...
```