



DnnOpenVinoDetector C++ library

v1.0.1

Table of contents

- [Overview](#)
- [Versions](#)
- [Library files](#)
- [Key features and capabilities](#)
- [Supported pixel formats](#)
- [Library principles](#)
- [DnnOpenVinoDetector class description](#)
 - [DnnOpenVinoDetector class declaration](#)
 - [getVersion method](#)
 - [initObjectDetector method](#)
 - [setParam method](#)
 - [getParam method](#)
 - [getParams method](#)
 - [executeCommand method](#)
 - [detect method](#)
 - [setMask method](#)
 - [decodeAndExecuteCommand method](#)
 - [encodeSetParamCommand method of ObjectDetector class](#)
 - [encodeCommand method of ObjectDetector class](#)
 - [decodeCommand method of ObjectDetector class](#)
- [Data structures](#)
 - [ObjectDetectorCommand enum](#)
 - [ObjectDetectorParam enum](#)
 - [Object structure](#)
- [ObjectDetectorParams class description](#)
 - [ObjectDetectorParams class declaration](#)
 - [Serialize object detector params](#)
 - [Deserialize object detector params](#)

- [Read params from JSON file and write to JSON file](#)
- [Build and connect to your project](#)
 - [On Windows](#)
 - [On Linux](#)
 - [Connect as source code](#)
- [Simple example](#)

Overview

DnnOpenVinoDetector C++ library version **1.0.1** is designed for automatic detection of objects on videos through the utilization of neural networks. The library is implemented in C++ (C++17 standard) and exclusively relies only on one third-party library, which is Intel's [OpenVINO™](#) runtime. The library supports various neural network models, if only model is supported by OpenVINO and has standard one-batch output (eg. yolov5). NN model can be specified in detector parameters, library inherits its interface from the [ObjectDetector](#) class, offering flexible and customizable parameters. The library provides simple interface.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	27.09.2023	First version.
1.0.1	04.01.2024	- Demo application updated. - Documentation updated.

Library files

The library is 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 to include third-party libraries
  ObjectDetector ----- files of ObjectDetector interface library
src ----- folder with library source code
  CMakeLists.txt ----- CMake file
  DnnOpenVinoDetector.h ----- main library header file
  DnnOpenVinoDetectorVersion.h ----- header file with library version
  DnnOpenVinoDetectorVersion.h.in --- file for CMake to generate version header
  DnnOpenVinoDetector.cpp ----- C++ implementation file
demo ----- folder for demo application files
  CMakeLists.txt ----- CMake file for demo app
  3rdaprtly ----- folder with 3rdparty libraries

```

```

CMakeLists.txt ----- CMake file for 3rdparty folder
SimpleFileDialog ----- file dialog 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

```

Demo application depends on open source [SimpleFileDialog](#), which provides file dialog function and open source [OpenCV](#) library to provide user interface.

Key features and capabilities

Table 2 - Key features and capabilities.

Parameter and feature	Description
Programming language	C++ (standard C++17) using the OpenVINO™ library (version 2022.3.1).
Supported OS	Compatible with any operating system that supports the C++ compiler (C++17 standard) and the OpenVINO library (version 2022.3.1).
Shape of detected objects	The library is capable of detecting moving objects of various shape. The minimum and maximum height and width of the objects to be detected are set by the user in the library parameters.
Supported pixel formats	RGB24, BGR24, GRAY, YUV24, YUYV, UYVY, NV12, NV21, YV12, YU12. The library uses the RGB24 format for video processing. If the pixel format of the image is different from RGB24, the library pre-converts the pixel formats to RGB24.
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 doesn't have a significant impact on the computational speed, because input images are resized to input network size (usually 640x640 pixels).
Coordinate system	The algorithm uses a window coordinate system with the zero point in the upper left corner of the video frame.
Calculation speed	The processing time per video frame depends mostly on loaded neural network model and also on the computing platform used. The processing time per video frame can be estimated with the demo application.
Type of algorithm for detection of objects	To detect different objects on current frame interface for obtaining neural network model was implemented. This interface relies on OpenVINO™ implementation and utilizes its features such as: pre-processing of input data, reading and launching neural network model and obtaining output results. Output data is cleaned up from overlapping boxes and correct types are assigned to output vector of objects.

Parameter and feature	Description
Discreteness of computation of coordinates	The library utilizes the object bounding box for each detected object. If boxes are overlapping and have the same object type, they are merged into one combined. This means discreteness of library is strongly depended on loaded neural network model.
Working conditions	The library is designed to function on a variety of devices. It is optimized for GPU hardware support, which can significantly enhance processing speed. The detector processes each frame independently, so camera movement does not impact the results.

Supported pixel formats

Frame library (included in **DnnOpenVinoDetector** library) contains **Fourcc** enum, which defines supported pixel formats (**Frame.h** file). **DnnOpenVinoDetector** library supports RAW pixel formats only. The library uses the **RGB24** format for video processing. If the pixel format of the image is different from **RGB24**, the library pre-converts the pixel formats to **RGB24**. **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.

pixel

R ₀₀	G ₀₀	B ₀₀	R ₀₁	G ₀₁	B ₀₁	R ₀₂	G ₀₂	B ₀₂	R ₀₃	G ₀₃	B ₀₃
R ₁₀	G ₁₀	B ₁₀	R ₁₁	G ₁₁	B ₁₁	R ₁₂	G ₁₂	B ₁₂	R ₁₃	G ₁₃	B ₁₃
R ₂₀	G ₂₀	B ₂₀	R ₂₁	G ₂₁	B ₂₁	R ₂₂	G ₂₂	B ₂₂	R ₂₃	G ₂₃	B ₂₃
R ₃₀	G ₃₀	B ₃₀	R ₃₁	G ₃₁	B ₃₁	R ₃₂	G ₃₂	B ₃₂	R ₃₃	G ₃₃	B ₃₃

RGB24

pixel

B ₀₀	G ₀₀	R ₀₀	B ₀₁	G ₀₁	R ₀₁	B ₀₂	G ₀₂	R ₀₂	B ₀₃	G ₀₃	R ₀₃
B ₁₀	G ₁₀	R ₁₀	B ₁₁	G ₁₁	R ₁₁	B ₁₂	G ₁₂	R ₁₂	B ₁₃	G ₁₃	R ₁₃
B ₂₀	G ₂₀	R ₂₀	B ₂₁	G ₂₁	R ₂₁	B ₂₂	G ₂₂	R ₂₂	B ₂₃	G ₂₃	R ₂₃
B ₃₀	G ₃₀	R ₃₀	B ₃₁	G ₃₁	R ₃₁	B ₃₂	G ₃₂	R ₃₂	B ₃₃	G ₃₃	R ₃₃

BGR24

pixel

Y ₀₀	U ₀₀	V ₀₀	Y ₀₁	U ₀₁	V ₀₁	Y ₀₂	U ₀₂	V ₀₂	Y ₀₃	U ₀₃	V ₀₃
Y ₁₀	U ₁₀	V ₁₀	Y ₁₁	U ₁₁	V ₁₁	Y ₁₂	U ₁₂	V ₁₂	Y ₁₃	U ₁₃	V ₁₃
Y ₂₀	U ₂₀	V ₂₀	Y ₂₁	U ₂₁	V ₂₁	Y ₂₂	U ₂₂	V ₂₂	Y ₂₃	U ₂₃	V ₂₃
Y ₃₀	U ₃₀	V ₃₀	Y ₃₁	U ₃₁	V ₃₁	Y ₃₂	U ₃₂	V ₃₂	Y ₃₃	U ₃₃	V ₃₃

YUV24

pixel

Y ₀₀	Y ₀₁	Y ₀₂	Y ₀₃
Y ₁₀	Y ₁₁	Y ₁₂	Y ₁₃
Y ₂₀	Y ₂₁	Y ₂₂	Y ₂₃
Y ₃₀	Y ₃₁	Y ₃₂	Y ₃₃

GRAY

macro pixel

Y ₀₀	U ₀₀	Y ₀₁	V ₀₀	Y ₀₂	U ₀₂	Y ₀₃	V ₀₂
Y ₁₀	U ₁₀	Y ₁₁	V ₁₀	Y ₁₂	U ₁₂	Y ₁₃	V ₁₂
Y ₂₀	U ₂₀	Y ₂₁	V ₂₀	Y ₂₂	U ₂₂	Y ₂₃	V ₂₂
Y ₃₀	U ₃₀	Y ₃₁	V ₃₀	Y ₃₂	U ₃₂	Y ₃₃	V ₃₂

YUYV

macro pixel

U ₀₀	Y ₀₀	V ₀₀	Y ₀₁	U ₀₂	Y ₀₂	Y ₀₃	V ₀₂
U ₁₀	Y ₁₀	V ₁₀	Y ₁₁	U ₁₂	Y ₁₂	Y ₁₃	V ₁₂
U ₂₀	Y ₂₀	V ₂₀	Y ₂₁	U ₂₂	Y ₂₂	Y ₂₃	V ₂₂
U ₃₀	Y ₃₀	V ₃₀	Y ₃₁	U ₃₂	Y ₃₂	Y ₃₃	V ₃₂

UYVY

macro pixel

Y ₀₀	Y ₀₁	Y ₀₂	Y ₀₃
Y ₁₀	Y ₁₁	Y ₁₂	Y ₁₃
Y ₂₀	Y ₂₁	Y ₂₂	Y ₂₃
Y ₃₀	Y ₃₁	Y ₃₂	Y ₃₃
U ₀₀	V ₀₀	U ₀₂	V ₀₂
U ₂₀	V ₂₀	U ₂₂	V ₂₂

NV12

macro pixel

Y ₀₀	Y ₀₁	Y ₀₂	Y ₀₃
Y ₁₀	Y ₁₁	Y ₁₂	Y ₁₃
Y ₂₀	Y ₂₁	Y ₂₂	Y ₂₃
Y ₃₀	Y ₃₁	Y ₃₂	Y ₃₃
V ₀₀	U ₀₀	V ₀₂	U ₀₂
V ₂₀	U ₂₀	V ₂₂	U ₂₂

NV21

macro pixel

Y ₀₀	Y ₀₁	Y ₀₂	Y ₀₃
Y ₁₀	Y ₁₁	Y ₁₂	Y ₁₃
Y ₂₀	Y ₂₁	Y ₂₂	Y ₂₃
Y ₃₀	Y ₃₁	Y ₃₂	Y ₃₃
U ₀₀	U ₀₂	U ₂₀	U ₂₂
V ₀₀	V ₀₂	V ₂₀	V ₂₂

YU12

macro pixel

Y ₀₀	Y ₀₁	Y ₀₂	Y ₀₃
Y ₁₀	Y ₁₁	Y ₁₂	Y ₁₃
Y ₂₀	Y ₂₁	Y ₂₂	Y ₂₃
Y ₃₀	Y ₃₁	Y ₃₂	Y ₃₃
V ₀₀	V ₀₂	V ₂₀	V ₂₂
U ₀₀	U ₀₂	U ₂₀	U ₂₂

YV12

Library Principles

The object detection feature within this library is seamlessly integrated with [OpenVINO™](#) runtime support, designed to facilitate efficient object detection based on neural network models. The library simplifies the process and provides a straightforward usage sequence for developers. The algorithm primarily consists of the following steps:

1. The library accepts input frames directly and sets proper inputs for compiling neural network model, eliminating the need for any preprocessing.
2. After model compilation, input frames are forwarded directly into the [OpenVINO™](#) pipeline, utilizing neural network models for object detection.
3. Computing results in output tensor, which is converted (according to detector parameters) to the final vector of objects.
4. Results store not only coordinates but also probability and type - which can be assigned to particular item, according to current network labels.

The library is available as source code only. To utilize the library as source code, developers must include the library's files into their project. The recommended usage sequence for the library is as follows:

1. **Integration:** Include the library files in your project.
2. **Initialization:** Create an instance of the `DnnOpenVinoDetector` C++ class for each camera or input source you wish to process. The library supports multiple instances for parallel camera processing.
3. **Customization:** If needed, you can customize the library's behavior by using the `setParam()` method. This allows you to adapt the library to specific requirements.
4. **Object Detection:** Create a `Frame` object to represent the input frame, and prepare a vector to store the detected objects.
5. **Detection Process:** Call the `detect(...)` method to initiate the object detection process.
6. **Object Retrieval:** Retrieve the detected objects by using the `getObjects()` method. The library provides a vector of `objects` containing information about the detected objects, such as their positions and attributes.

DnnOpenVinoDetector class description

DnnOpenVinoDetector class declaration

`DnnOpenVinoDetector.h` file contains `DnnOpenVinoDetector` class declaration. `DnnOpenVinoDetector` class inherits interface from [ObjectDetector](#) interface class. Class declaration:

```
class DnnOpenVinoDetector : public ObjectDetector
{
public:
    /// Get string of current library version.
    static std::string getVersion();
```

```

/// Init object detector.
bool initObjectDetector(ObjectDetectorParams& params) override;

/// Set object detector param.
bool setParam(ObjectDetectorParam id, float value) override;

/// Get object detector param value.
float getParam(ObjectDetectorParam id) override;

/// Get object detector params structure.
void getParams(ObjectDetectorParams& params) override;

/// Get list of objects.
std::vector<Object> getObjects() override;

/// Execute command.
bool executeCommand(ObjectDetectorCommand id) override;

/// Perform detection.
bool detect(cr::video::Frame& frame) override;

/// Set detection mask.
bool setMask(cr::video::Frame mask) override;

/// Decode command and execute command.
bool decodeAndExecuteCommand(uint8_t* data, int size) override;
}

```

getVersion method

getVersion() method returns string of current version of **DnnOpenVinoDetector** class. Method declaration:

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

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

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

Console output:

```
DnnOpenVinoDetector version: 1.0.1
```

initObjectDetector method

initObjectDetector(...) method initializes object detector. Inside this method also reading network model takes place, because of that it can consume much time when loading big neural network models. Method declaration:

```
bool initObjectDetector(ObjectDetectorParams& params) override;
```

Parameter	Value
params	Object detector parameters class. Object detector should initialize all parameters listed in ObjectDetectorParams . The library takes into account only following parameters from ObjectDetectorParams class: minObjectWidth (Default value 4) maxObjectWidth (Default value 128) minObjectHeight (Default value 4) maxObjectHeight (Default value 128) minDetectionPropability(Default value 0.5f) type(Default value 0) If particular parameter out of valid range the library will set default values automatically.

Returns: TRUE if the object detector was initialized or FALSE if not.

setParam method

setParam(...) method designed to set new DnnOpenVinoDetector object parameter value. Method declaration:

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

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

Returns: TRUE if the parameter was set or FALSE if not (not supported or out of valid range).

getParam method

getParam(...) method designed to obtain object detector parameter value. Method declaration:

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


Parameter	Description
id	Parameter ID according to ObjectDetectorParam enum.

Returns: parameter value or -1.0f if the parameter is not supported by DnnOpenVinoDetector.

getParams method

getParams(...) method designed to obtain object detector params structures as well as a list of detected objects. Method declaration:

```
void getParams(ObjectDetectorParams& params) override;
```

Parameter	Description
params	Object detector params object (ObjectDetectorParams). The library supports not all parameters from ObjectDetector interface. If the library doesn't support particular parameter it will be immutable and will have default value.

getObjects method

getObjects() method designed to obtain list of detected objects. User can obtain list of detected objects via **getParams(...)** method as well. Method declaration:

```
std::vector<Object> getObjects() override;
```

Returns: list of detected objects (see **Object** class description). If no objects are detected, the list will be empty.

executeCommand method

executeCommand(...) method designed to execute object detector command. Method declaration:

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

Parameter	Description
id	Command ID according to ObjectDetectorCommand enum.

Returns: TRUE if the command was executed or FALSE if not (only if command ID not valid).

detect method

detect(...) method designed to perform detection algorithm. Method declaration:

```
bool detect(cr::video::Frame& frame) override;
```

Parameter	Description
frame	Video frame for processing. Object detector processes only RAW pixel formats (BGR24, RGB24, GRAY, YUYV24, YUYV, UYVY, NV12, NV21, YV12, YU12, see Frame class description). Size of frame should be from 24x24 to 8192x8192.

Returns: TRUE if the video frame was processed FALSE if not. *Note:* If object detector disabled (see **ObjectDetectorParam** enum description) the method returns TRUE.

setMask method

setMask(...) method designed to set detection mask. The user can disable detection 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 detection of objects in the corresponding place of video frames. Any other mask pixel value other than 0 allows detection of objects at the corresponding location of video frames. The mask is used for detection 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) override;
```

Parameter	Description
mask	Image of detection mask. Should have GRAY (preferable), NV12, NV21, YV12 or YU12 pixel format (see Frame). Detector omits image segments, where detection mask pixels' values equal 0. Mask can have any resolution. If resolution of mask (width and height) is not equal to video frame resolution, the library will scale this mask up to original processed video resolution.

Returns: TRUE if the the mask accepted or FALSE if not (not valid pixel format or empty).

decodeAndExecuteCommand method

decodeAndExecuteCommand(...) method decodes and executes command on object detector side. **decodeAndExecuteCommand(...)** is thread-safe method.

Method declaration:

```
bool decodeAndExecuteCommand(uint8_t* data, int size) override;
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM or 7 bytes for COMMAND.

Returns: TRUE if command decoded (SET_PARAM or COMMAND) and executed (action command or set param command).

encodeSetParamCommand method of ObjectDetector class

encodeSetParamCommand(...) static method of the [ObjectDetector](#) interface designed to encode command to change any parameter for remote object detector (including motion detectors). To control object detector remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **ObjectDetector** class contains static methods for encoding the control command. The **ObjectDetector** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND).

encodeSetParamCommand(...) designed to encode SET_PARAM command. Method declaration:

```
static void encodeSetParamCommand(uint8_t* data, int& size, ObjectDetectorParam id, float value);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.
id	Parameter ID according to ObjectDetectorParam enum.
value	Parameter value. Value depends on parameter ID.

SET_PARAM command format:

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	Major	Major version of ObjectDetector class.
2	Minor	Minor version of ObjectDetector class.
3	id	Parameter ID int32_t in Little-endian format.
4	id	Parameter ID int32_t in Little-endian format.
5	id	Parameter ID int32_t in Little-endian format.
6	id	Parameter ID int32_t in Little-endian format.
7	value	Parameter value float in Little-endian format.

Byte	Value	Description
8	value	Parameter value float in Little-endian format.
9	value	Parameter value float in Little-endian format.
10	value	Parameter value float in Little-endian format.

encodeSetParamCommand(...) is static and used without **ObjectDetector** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Random parameter value.
float outValue = (float)(rand() % 20);
// Encode command.
ObjectDetector::encodeSetParamCommand(data, size, ObjectDetectorParam::MIN_OBJECT_WIDTH,
outValue);
```

encodeCommand method of ObjectDetector class

encodeCommand(...) static method of the [ObjectDetector](#) interface designed to encode command for remote object detector (including motion detectors). To control object detector remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **ObjectDetector** class contains static methods for encoding the control command. The **ObjectDetector** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeCommand(...)** designed to encode COMMAND (action command). Method declaration:

```
static void encodeCommand(uint8_t* data, int& size, ObjectDetectorCommand id);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.
id	Command ID according to ObjectDetectorCommand enum.

COMMAND format:

Byte	Value	Description
0	0x00	COMMAND header value.
1	Major	Major version of ObjectDetector class.
2	Minor	Minor version of ObjectDetector class.

Byte	Value	Description
3	id	Command ID int32_t in Little-endian format.
4	id	Command ID int32_t in Little-endian format.
5	id	Command ID int32_t in Little-endian format.
6	id	Command ID int32_t in Little-endian format.

encodeCommand(...) is static and used without **ObjectDetector** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Encode command.
ObjectDetector::encodeCommand(data, size, ObjectDetectorCommand::RESET);
```

decodeCommand method of ObjectDetector class

decodeCommand(...) static method of the [ObjectDetector](#) interface designed to decode command on object detector side (edge device). Method declaration:

```
static int decodeCommand(uint8_t* data, int size, ObjectDetectorParam& paramId,
ObjectDetectorCommand& commandId, float& value);
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Should be 11 bytes.
paramId	Parameter ID according to ObjectDetectorParam enum. After decoding SET_PARAM command the method will return parameter ID.
commandId	Command ID according to ObjectDetectorCommand enum. After decoding COMMAND the method will return command ID.
value	Parameter value (after decoding SET_PARAM command).

Returns: **0** - in case decoding COMMAND, **1** - in case decoding SET_PARAM command or **-1** in case errors.

Data structures

ObjectDetectorCommand enum

Enum declaration:

```
enum class ObjectDetectorCommand
{
    /// Reset.
    RESET = 1,
    /// Enable.
    ON,
    /// Disable.
    OFF
};
```

Table 4 - Object detector commands description. Some commands maybe unsupported by particular object detector class.

Command	Description
RESET	Reset algorithm. Clears the list of detected objects and resets all internal filters.
ON	Enable object detector. If the detector is not activated, frame processing is not performed - the list of detected objects will always be empty.
OFF	Disable object detector. If the detector is not activated, frame processing is not performed - the list of detected objects will always be empty.

ObjectDetectorParam enum

Enum declaration:

```
enum class ObjectDetectorParam
{
    /// Logging mode. values: 0 - Disable, 1 - Only file,
    /// 2 - Only terminal (console), 3 - File and terminal (console).
    LOG_MODE = 1,
    /// Frame buffer size. Depends on implementation.
    FRAME_BUFFER_SIZE,
    /// Minimum object width to be detected, pixels. To be detected object's
    /// width must be >= MIN_OBJECT_WIDTH.
    MIN_OBJECT_WIDTH,
    /// Maximum object width to be detected, pixels. To be detected object's
    /// width must be <= MAX_OBJECT_WIDTH.
    MAX_OBJECT_WIDTH,
    /// Minimum object height to be detected, pixels. To be detected object's
    /// height must be >= MIN_OBJECT_HEIGHT.
    MIN_OBJECT_HEIGHT,
    /// Maximum object height to be detected, pixels. To be detected object's
    /// height must be <= MAX_OBJECT_HEIGHT.
    MAX_OBJECT_HEIGHT,
    /// Minimum object's horizontal speed to be detected, pixels/frame. To be
    /// detected object's horizontal speed must be >= MIN_X_SPEED.
};
```

```

MIN_X_SPEED,
/// Maximum object's horizontal speed to be detected, pixels/frame. To be
/// detected object's horizontal speed must be <= MAX_X_SPEED.
MAX_X_SPEED,
/// Minimum object's vertical speed to be detected, pixels/frame. To be
/// detected object's vertical speed must be >= MIN_Y_SPEED.
MIN_Y_SPEED,
/// Maximum object's vertical speed to be detected, pixels/frame. To be
/// detected object's vertical speed must be <= MAX_Y_SPEED.
MAX_Y_SPEED,
/// Probability threshold from 0 to 1. To be detected object detection
/// probability must be >= MIN_DETECTION_PROPABILITY.
MIN_DETECTION_PROPABILITY,
/// Horizontal track detection criteria, frames. By default shows how many
/// frames the objects must move in any(+/-) horizontal direction to be
/// detected.
X_DETECTION_CRITERIA,
/// Vertical track detection criteria, frames. By default shows how many
/// frames the objects must move in any(+/-) vertical direction to be
/// detected.
Y_DETECTION_CRITERIA,
/// Track reset criteria, frames. By default shows how many
/// frames the objects should be not detected to be excluded from results.
RESET_CRITERIA,
/// Detection sensitivity. Depends on implementation. Default from 0 to 1.
SENSITIVITY,
/// Frame scaling factor for processing purposes. Reduce the image size by
/// scaleFactor times horizontally and vertically for faster processing.
SCALE_FACTOR,
/// Num threads. Number of threads for parallel computing.
NUM_THREADS,
/// Processing time for last frame, mks.
PROCESSING_TIME_MKS,
/// Algorithm type. Depends on implementation.
TYPE,
/// Mode. Default: 0 - Off, 1 - On.
MODE,
/// Custom parameter. Depends on implementation.
CUSTOM_1,
/// Custom parameter. Depends on implementation.
CUSTOM_2,
/// Custom parameter. Depends on implementation.
CUSTOM_3
};

```

Table 5 - DnnOpenVinoDetector class params description (from [ObjectDetector](#) interface class). Some params are not supported by DnnOpenVinoDetector class.

Parameter	Access	Description
LOG_MODE	read / write	Not used. Can have any value.
FRAME_BUFFER_SIZE	read / write	Not used. Can have any value.

Parameter	Access	Description
MIN_OBJECT_WIDTH	read / write	Minimum object width to be detected, pixels. Valid values from 1 to 8192. Must be < MAX_OBJECT_WIDTH. To be detected object's width must be >= MIN_OBJECT_WIDTH. Default value is 4.
MAX_OBJECT_WIDTH	read / write	Maximum object width to be detected, pixels. Valid values from 1 to 8192. Must be > MIN_OBJECT_WIDTH. To be detected object's width must be <= MAX_OBJECT_WIDTH. Default value is 128.
MIN_OBJECT_HEIGHT	read / write	Minimum object height to be detected, pixels. Valid values from 1 to 8192. Must be < MAX_OBJECT_HEIGHT. To be detected object's height must be >= MIN_OBJECT_HEIGHT. Default value is 4.
MAX_OBJECT_HEIGHT	read / write	Maximum object height to be detected, pixels. Valid values from 1 to 8192. Must be > MIN_OBJECT_HEIGHT. To be detected object's height must be <= MAX_OBJECT_HEIGHT. Default value is 128.
MIN_X_SPEED	read / write	Not used. Can have any value.
MAX_X_SPEED	read / write	Not used. Can have any value.
MIN_Y_SPEED	read / write	Not used. Can have any value.
MAX_Y_SPEED	read / write	Not used. Can have any value.
MIN_DETECTION_PROBABILITY	read / write	Defines threshold for object detection probability. Only objects with probability greater than MIN_DETECTION_PROBABILITY will be detected. Can have any values from 0 to 1.
X_DETECTION_CRITERIA	read / write	Not used. Can have any value.
Y_DETECTION_CRITERIA	read / write	Not used. Can have any value.
RESET_CRITERIA	read / write	Not used. Can have any value.
SENSITIVITY	read / write	Not used. Can have any value.
SCALE_FACTOR	read / write	Not used. Can have any value.

Parameter	Access	Description
NUM_THREADS	read / write	Not used. Can have any value.
PROCESSING_TIME_MKS	read only	Not used. Can have any value.
TYPE	read / write	Type defines kind of device for neural network model computation. Default 0 - CPU, 1 - integrated GPU (GPU.0), 2 - separate GPU (GPU.1) 3 - both GPUs (MULTI mode). More information - OpenVinoDevices .
MODE	read / write	Mode. Default: 0 - Off, 1 - On. If the detector is not activated, frame processing is not performed - the list of detected objects will always be empty.
CUSTOM_1	read / write	Not used. Can have any value.
CUSTOM_2	read / write	Not used. Can have any value.
CUSTOM_3	read / write	Not used. Can have any value.

Object structure

Object structure is used to describe detected object. Object structure declared in **ObjectDetector.h** file. Structure declaration:

```
typedef struct Object
{
    /// Object ID. Must be uniques for particular object.
    int id{0};
    /// Frame ID. Must be the same as frame ID of processed video frame.
    int frameId{0};
    /// Object type. Depends on implementation.
    int type{0};
    /// Object rectangle width, pixels.
    int width{0};
    /// Object rectangle height, pixels.
    int height{0};
    /// Object rectangle top-left horizontal coordinate, pixels.
    int x{0};
    /// Object rectangle top-left vertical coordinate, pixels.
    int y{0};
    /// Horizontal component of object velocity, +-pixels/frame.
    float vx{0.0f};
    /// Vertical component of object velocity, +-pixels/frame.
    float vy{0.0f};
    /// Detection probability from 0 (minimum) to 1 (maximum).
    float p{0.0f};
};
```

```
} Object;
```

Table 6 - Object structure fields description.

Field	Type	Description
id	int	Object ID on a current frame.
frameId	int	Frame ID. Will be the same as frame ID of processed video frame.
type	int	Object type according to probability for particular label that was returned from neural network model inference output.
width	int	Object rectangle width, pixels. Must be MIN_OBJECT_WIDTH <= width <= MAX_OBJECT_WIDTH (see ObjectDetectorParam enum description).
height	int	Object rectangle height, pixels. Must be MIN_OBJECT_HEIGHT <= height <= MAX_OBJECT_HEIGHT (see ObjectDetectorParam enum description).
x	int	Object rectangle top-left horizontal coordinate, pixels.
y	int	Object rectangle top-left vertical coordinate, pixels.
vX	float	Not used. Will have value 0.0f.
vY	float	Not used. Will have value 0.0f.
p	float	Probability value for object label, NOT probability of whole set of labels.

ObjectDetectorParams class description

ObjectDetectorParams class declaration

ObjectDetectorParams class used for object detector initialization (**initObjectDetector(...)** method) or to get all actual params (**getParams()** method) including list of detected objects. Also **ObjectDetectorParams** provides 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 ObjectDetectorParams
{
public:
    /// Init string. Depends on implementation.
    std::string initString{""};
    /// Logging mode. values: 0 - Disable, 1 - Only file,
    /// 2 - Only terminal (console), 3 - File and terminal (console).
    int logMode{0};
    /// Frame buffer size. Depends on implementation.
    int framebufferSize{1};
    /// Minimum object width to be detected, pixels. To be detected object's
    /// width must be >= minObjectwidth.
    int minObjectwidth{4};
```

```

/// Maximum object width to be detected, pixels. To be detected object's
/// width must be <= maxObjectWidth.
int maxObjectWidth{128};
/// Minimum object height to be detected, pixels. To be detected object's
/// height must be >= minObjectHeight.
int minObjectHeight{4};
/// Maximum object height to be detected, pixels. To be detected object's
/// height must be <= maxObjectHeight.
int maxObjectHeight{128};
/// Minimum object's horizontal speed to be detected, pixels/frame. To be
/// detected object's horizontal speed must be >= minXSpeed.
float minXSpeed{0.0f};
/// Maximum object's horizontal speed to be detected, pixels/frame. To be
/// detected object's horizontal speed must be <= maxXSpeed.
float maxXSpeed{30.0f};
/// Minimum object's vertical speed to be detected, pixels/frame. To be
/// detected object's vertical speed must be >= minYSpeed.
float minYSpeed{0.0f};
/// Maximum object's vertical speed to be detected, pixels/frame. To be
/// detected object's vertical speed must be <= maxYSpeed.
float maxYSpeed{30.0f};
/// Probability threshold from 0 to 1. To be detected object detection
/// probability must be >= minDetectionProbability.
float minDetectionProbability{0.5f};
/// Horizontal track detection criteria, frames. By default shows how many
/// frames the objects must move in any(+/-) horizontal direction to be
/// detected.
int xDetectionCriteria{1};
/// Vertical track detection criteria, frames. By default shows how many
/// frames the objects must move in any(+/-) vertical direction to be
/// detected.
int yDetectionCriteria{1};
/// Track reset criteria, frames. By default shows how many
/// frames the objects should be not detected to be excluded from results.
int resetCriteria{1};
/// Detection sensitivity. Depends on implementation. Default from 0 to 1.
float sensitivity{0.04f};
/// Frame scaling factor for processing purposes. Reduce the image size by
/// scaleFactor times horizontally and vertically for faster processing.
int scaleFactor{1};
/// Num threads. Number of threads for parallel computing.
int numThreads{1};
/// Processing time for last frame, mks.
int processingTimeMks{0};
/// Algorithm type. Depends on implementation.
int type{0};
/// Mode. Default: false - Off, on - On.
bool enable{true};
/// 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};
/// List of detected objects.

```

```

std::vector<Object> objects;

JSON_READABLE(ObjectDetectorParams, initString, logMode, frameBufferSize,
               minObjectWidth, maxObjectWidth, minObjectHeight, maxObjectHeight,
               minXSpeed, maxXSpeed, minYSpeed, maxYSpeed, minDetectionProbability,
               xDetectionCriteria, yDetectionCriteria, resetCriteria, sensitivity,
               scaleFactor, numThreads, type, enable, custom1, custom2, custom3);
/**
 * @brief operator =
 * @param src Source object.
 * @return ObjectDetectorParams object.
 */
ObjectDetectorParams& operator= (const ObjectDetectorParams& src);
/**
 * @brief Encode params. Method doesn't encode initString.
 * @param data Pointer to data buffer.
 * @param size Size of data.
 * @param mask Pointer to parameters mask.
 */
void encode(uint8_t* data, int& size,
            ObjectDetectorParamsMask* mask = nullptr);
/**
 * @brief Decode params. Method doesn't decode initString;
 * @param data Pointer to data.
 * @return TRUE is params decoded or FALSE if not.
 */
bool decode(uint8_t* data);
};

```

Table 7 - ObjectDetectorParams class fields description. Some params may be unsupported by DnnOpenVinoDetector class.

Field	Type	Description
initString	string	Has to include path to neural network model that is supported by OpenVino and has standard one-batch layout. Also has to include dimension (width;height) of neural network input images, everything separated by semicolon. E.g.: "./model.onnx;640;640". If neural network model consists of 2 files it should be as this exmaple: "./model.xml;./model.bin;256;480".
logMode	int	Not used. Can have any value.
frameBufferSize	int	Not used. Can have any value.
minObjectWidth	int	Minimum object width to be detected, pixels. Valid values from 1 to 8192. Must be < maxObjectWidth. To be detected object's width must be >= minObjectWidth. Default value is 4.

Field	Type	Description
maxObjectWidth	int	Maximum object width to be detected, pixels. Valid values from 1 to 8192. Must be > minObjectWidth. To be detected object's width must be <= maxObjectWidth. Default value is 128.
minObjectHeight	int	Minimum object height to be detected, pixels. Valid values from 1 to 8192. Must be < maxObjectHeight. To be detected object's height must be >= minObjectHeight. Default value is 4.
maxObjectHeight	int	Maximum object height to be detected, pixels. Valid values from 1 to 8192. Must be > minObjectHeight. To be detected object's height must be <= maxObjectHeight. Default value is 128.
minXSpeed	float	Not used. Can have any value.
maxXSpeed	float	Not used. Can have any value.
minYSpeed	float	Not used. Can have any value.
maxYSpeed	float	Not used. Can have any value.
minDetectionProbability	float	Defines threshold for object detection probability. Only objects with probability greater than MIN_DETECTION_PROBABILITY will be detected. Can have any values from 0 to 1.
xDetectionCriteria	int	Not used. Can have any value.
yDetectionCriteria	int	Not used. Can have any value.
resetCriteria	int	Not used. Can have any value.
sensitivity	float	Not used. Can have any value.
scaleFactor	int	Not used. Can have any value.
numThreads	int	Not used. Can have any value.
processingTimeMks	int	Not used. Can have any value.
type	int	Type defines kind of device for neural network model computation. Default 0 - CPU, 1 - integrated GPU (GPU.0), 2 - separate GPU (GPU.1) 3 - both GPUs (MULTI mode). More information - OpenVinoDevices .
enable	bool	Mode: false - Off, true - On. If the detector is not activated, frame processing is not performed - the list of detected objects will always be empty.
custom1	float	Not used. Can have any value.
custom2	float	Not used. Can have any value.

Field	Type	Description
custom3	float	Not used. Can have any value.
objects	std::vector	List of detected objects.

Note: *ObjectDetectorParams* class fields listed in Table 7 **must** reflect params set/get by methods *setParam(...)* and *getParam(...)*.

Serialize object detector params

ObjectDetectorParams class provides method **encode(...)** to serialize object detector params (fields of *ObjectDetectorParams* class, see Table 5). Serialization of object detector params necessary in case when you need to send params via communication channels. Method provides options to exclude particular parameters from serialization. To do this method inserts binary mask (3 bytes) where each bit represents particular parameter and **decode(...)** method recognizes it. Method doesn't encode *initString*. Method declaration:

```
void encode(uint8_t* data, int dataBufferSize, int& size, ObjectDetectorParamsMask* mask = nullptr);
```

Parameter	Value
data	Pointer to data buffer. Buffer size should be at least 99 bytes.
dataBufferSize	Size of data buffer. If the data buffer size is not large enough to serialize all detected objects (40 bytes per object), not all objects will be included in the data.
size	Size of encoded data. 99 bytes by default.
mask	Parameters mask - pointer to ObjectDetectorParamsMask structure. ObjectDetectorParamsMask (declared in <i>ObjectDetector.h</i> file) determines flags for each field (parameter) declared in ObjectDetectorParams class. If the user wants to exclude any parameters from serialization, he can put a pointer to the mask. If the user wants to exclude a particular parameter from serialization, he should set the corresponding flag in the <i>ObjectDetectorParamsMask</i> structure.

ObjectDetectorParamsMask structure declaration:

```
typedef struct ObjectDetectorParamsMask
{
    bool logMode{true};
    bool frameBufferSize{true};
    bool minObjectWidth{true};
    bool maxObjectWidth{true};
    bool minObjectHeight{true};
    bool maxObjectHeight{true};
    bool minXSpeed{true};
    bool maxXSpeed{true};
    bool minYSpeed{true};
    bool maxYSpeed{true};
}
```

```

bool minDetectionProbability{true};
bool xDetectionCriteria{true};
bool yDetectionCriteria{true};
bool resetCriteria{true};
bool sensitivity{true};
bool scaleFactor{true};
bool numThreads{true};
bool processingTimeMks{true};
bool type{true};
bool enable{true};
bool custom1{true};
bool custom2{true};
bool custom3{true};
bool objects{true};
} ObjectDetectorParamsMask;

```

Example without parameters mask:

```

// Prepare random params.
ObjectDetectorParams in;
in.logMode = rand() % 255;
in.objects.clear();
for (int i = 0; i < 5; ++i)
{
    Object obj;
    obj.id = rand() % 255;
    obj.type = rand() % 255;
    obj.width = rand() % 255;
    obj.height = rand() % 255;
    obj.x = rand() % 255;
    obj.y = rand() % 255;
    obj.vx = rand() % 255;
    obj.vy = rand() % 255;
    obj.p = rand() % 255;
    in.objects.push_back(obj);
}

// Encode data.
uint8_t data[1024];
int size = 0;
in.encode(data, size);
cout << "Encoded data size: " << size << " bytes" << endl;

```

Example with parameters mask:

```

// Prepare random params.
ObjectDetectorParams in;
in.logMode = rand() % 255;
in.objects.clear();
for (int i = 0; i < 5; ++i)
{
    Object obj;
    obj.id = rand() % 255;
    obj.type = rand() % 255;

```

```

obj.width = rand() % 255;
obj.height = rand() % 255;
obj.x = rand() % 255;
obj.y = rand() % 255;
obj.vX = rand() % 255;
obj.vY = rand() % 255;
obj.p = rand() % 255;
in.objects.push_back(obj);
}

// Prepare mask.
ObjectDetectorParamsMask mask;
mask.logMode = false;

// Encode data.
uint8_t data[1024];
int size = 0;
in.encode(data, size, &mask)
cout << "Encoded data size: " << size << " bytes" << endl;

```

Deserialize object detector params

ObjectDetectorParams class provides method **decode(...)** to deserialize params (fields of **ObjectDetectorParams** class, see Table 5). Deserialization of params necessary in case when you need to receive params via communication channels. Method automatically recognizes which parameters were serialized by **encode(...)** method. Method doesn't decode **initString**. Method declaration:

```
bool decode(uint8_t* data);
```

Parameter	Value
data	Pointer to encode data buffer.

Returns: TRUE if data decoded (deserialized) or FALSE if not.

Example:

```

// Prepare random params.
ObjectDetectorParams in;
in.logMode = rand() % 255;
for (int i = 0; i < 5; ++i)
{
    object obj;
    obj.id = rand() % 255;
    obj.type = rand() % 255;
    obj.width = rand() % 255;
    obj.height = rand() % 255;
    obj.x = rand() % 255;
    obj.y = rand() % 255;
    obj.vX = rand() % 255;
    obj.vY = rand() % 255;
    obj.p = rand() % 255;
}

```



```

        in.objects.push_back(obj);
    }

    // Encode data.
    uint8_t data[1024];
    int size = 0;
    in.encode(data, size);
    cout << "Encoded data size: " << size << " bytes" << endl;

    // Decode data.
    ObjectDetectorParams out;
    if (!out.decode(data))
    {
        cout << "Can't decode data" << endl;
        return false;
    }
}

```

Read params from JSON file and write to JSON file

ObjectDetector library depends on **ConfigReader** library which provides method to read params from JSON file and to write params to JSON file. Example of writing and reading params to JSON file:

```

// Prepare random params.
ObjectDetectorParams in;
in.logMode = rand() % 255;
in.objects.clear();
for (int i = 0; i < 5; ++i)
{
    Object obj;
    obj.id = rand() % 255;
    obj.type = rand() % 255;
    obj.width = rand() % 255;
    obj.height = rand() % 255;
    obj.x = rand() % 255;
    obj.y = rand() % 255;
    obj.vX = rand() % 255;
    obj.vY = rand() % 255;
    obj.p = rand() % 255;
    in.objects.push_back(obj);
}

// Write params to file.
cr::utils::ConfigReader inConfig;
inConfig.set(in, "ObjectDetectorParams");
inConfig.writeToFile("ObjectDetectorParams.json");

// Read params from file.
cr::utils::ConfigReader outConfig;
if (!outConfig.readFromFile("ObjectDetectorParams.json"))
{
    cout << "Can't open config file" << endl;
    return false;
}

```

```
ObjectDetectorParams out;
if(!outConfig.get(out, "ObjectDetectorParams"))
{
    cout << "Can't read params from file" << endl;
    return false;
}
```

ObjectDetectorParams.json will look like:

```
{
  "ObjectDetectorParams": {
    "custom1": 57.0,
    "custom2": 244.0,
    "custom3": 68.0,
    "enable": false,
    "frameBufferSize": 200,
    "initString": "sfsfsfsfsf",
    "logMode": 111,
    "maxObjectHeight": 103,
    "maxObjectWidth": 199,
    "maxXSpeed": 104.0,
    "maxYSpeed": 234.0,
    "minDetectionProbability": 53.0,
    "minObjectHeight": 191,
    "minObjectWidth": 149,
    "minXSpeed": 213.0,
    "minYSpeed": 43.0,
    "numThreads": 33,
    "resetCriteria": 62,
    "scaleFactor": 85,
    "sensitivity": 135.0,
    "type": 178,
    "xDetectionCriteria": 224,
    "yDetectionCriteria": 199
  }
}
```

Build and connect to your project

On Windows

Start with cloning **DnnOpenVinoDetector** to your local directory on Windows:

```
cd DnnOpenVinoDetector
git submodule update --init --recursive
```

Necessary step before building **DnnOpenVinoDetector** library is to download **OpenVino** runtime library. Current tested and working perfectly with **DnnOpenVinoDetector** library **OpenVino** runtime library version is **2022.3.1**. [DOCS](#)

Steps for connecting OpenVino to **DnnOpenVinoDetector** project:

1. Create directory "C:\Program Files (x86)\Intel".
2. Download OpenVINO compressed library from this link -> [LINK](#)
3. Extract .zip file to "C:\Program Files (x86)\Intel" and rename extracted folder to 'openvino'.
4. Add paths to Path in "Edit the system environment variables" window:

```
- C:\Program Files (x86)\Intel\openvino\runtime\bin\intel64\Release
- C:\Program Files (x86)\Intel\openvino\runtime\bin\intel64\Debug
- C:\Program Files (x86)\Intel\openvino\runtime\3rdparty\tbb\bin
- C:\Program Files (x86)\Intel\openvino\runtime\3rdparty\hddl\bin
```

5. Run cmake command with pointing where OpenVino CMake is placed and build whole solution:

```
mkdir build
cd build
cmake .. -D CMAKE_PREFIX_PATH=C:/Program Files (x86)/Intel/openvino/runtime/cmake
make
```

On Linux

Start with copying **DnnOpenVinoDetector** to your local directory on Linux:

```
cd DnnOpenVinoDetector
git submodule update --init --recursive
```

Necessary step before building **DnnOpenVinoDetector** library is to download **OpenVino** runtime library. Current tested and working perfectly with **DnnOpenVinoDetector** library **OpenVino** runtime library version is **2022.3.1**. [DOCS](#)

1. Make sure you have the latest updates.

```
sudo apt-get update
sudo apt-get upgrade
```

2. Get OpenVINO 2022.0 and place it in opt/intel directory.

```
cd <user_home>/Downloads
sudo apt-get install curl
curl -L
https://storage.openvino toolkit.org/repositories/openvino/packages/2022.3.1/linux/l_o
penvino_toolkit_ubuntu20_2022.3.1.9227.cf2c7da5689_x86_64.tgz --output
openvino_2022.3.1.tgz
tar -xf openvino_2022.3.1.tgz
sudo mkdir /opt/intel
sudo mv l_openvino_toolkit_ubuntu20_2022.3.1.9227.cf2c7da5689_x86_64
openvino_2022.3.1
sudo mv openvino_2022.3.1 /opt/intel/
```

3. Install required system dependencies on Linux.

```
cd /opt/intel/opencvino_2022.3.1
sudo -E ./install_dependencies/install_opencvino_dependencies.sh
```

4. Configure the environment.

```
source /opt/intel/opencvino_2022.3.1/setupvars.sh
```

5. Add libs for GPU devices.

```
sudo apt-get install -y ocl-icd-libopencl1 intel-opencl-icd
```

6. Run cmake command with pointing where OpenVino cmake is placed and build whole solution:

```
cd <project_dir>/DnnOpenVinoDetector/
mkdir build
cd build
cmake .. -D CMAKE_PREFIX_PATH=/opt/intel/opencvino_2022.3.1/runtime/cmake
make
```

Connect as source code

If you want to connect **DnnOpenVinoDetector** 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 **DnnOpenVinoDetector** as git submodule by commands (only if you have access to GitHub repository, if not copy repository folder):

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

In your repository folder, a new **3rdparty/DnnOpenVinoDetector** folder will be created, which contains files from **DnnOpenVinoDetector** repository along with its subrepository **ObjectDetector** and its subrepositories **Frame** and **ConfigReader**. If you don't have access to GitHub repository, copy **DnnOpenVinoDetector** repository folder to **3rdparty** folder to your repository. Remember to add OpenVino library as above. The new structure of your repository will be as follows:

```

CMakeLists.txt
src
  CMakeList.txt
  yourLib.h
  yourLib.cpp
3rdparty
  DnnOpenVinoDetector

```

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(${PARENT}_SUBMODULE_DNN_OPENVINO_DETECTOR      ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_DNN_OPENVINO_DETECTOR)
  SET(${PARENT}_DNN_OPENVINO_DETECTOR              ON  CACHE BOOL "" FORCE)
  SET(${PARENT}_DNN_OPENVINO_DETECTOR_TEST         OFF CACHE BOOL "" FORCE)
  SET(${PARENT}_DNN_OPENVINO_DETECTOR_DEMO_APP     OFF CACHE BOOL "" FORCE)
  SET(${PARENT}_DNN_OPENVINO_DETECTOR_EXAMPLE     OFF CACHE BOOL "" FORCE)
endif()

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

```

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

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

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

Done!

Simple example

A simple application shows how to use the DnnOpenVinoDetector library. The application opens a video file "test.mp4" and copies the video frame data into an object of the Frame class and performs objects detection.

```
#include <opencv2/opencv.hpp>
#include "DnnOpenVinoDetector.h"

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

    // Create and init detector.
    cr::detector::DnnOpenVinoDetector detector;
    cr::detector::ObjectDetectorParams params;
    params.initString = "./yolov7s.onnx;640;640";
    params.maxObjectHeight = 96;
    params.maxObjectWidth = 96;
    params.minObjectHeight = 4;
    params.minObjectWidth = 4;
    params.type = 1;
    detector.initObjectDetector(params);

    // Create frames.
    cv::Mat frameBgrOpenCv;

    // Main loop.
```

```

while (true)
{
    // Capture next video frame.
    videoSource >> frameBgrOpenCv;
    if (frameBgrOpenCv.empty())
    {
        // If we have video file we can set initial position to replay.
        detector.executeCommand(cr::detector::ObjectDetectorCommand::RESET);
        videoSource.set(cv::CAP_PROP_POS_FRAMES, 0);
        continue;
    }

    // Copy frame data from OpenCv frame to Frame.
    cr::video::Frame frameBgr;
    frameBgr.fourcc = cr::video::Fourcc::BGR24;
    frameBgr.width = frameBgrOpenCv.size().width;
    frameBgr.height = frameBgrOpenCv.size().height;
    frameBgr.size = frameBgr.width * frameBgr.height * 3;
    frameBgr.data = frameBgrOpenCv.data;

    // Detect objects.
    detector.detect(frameBgr);

    // Get list of objects.
    std::vector<cr::detector::Object> objects = detector.getObjects();

    // Draw detected objects.
    for (int n = 0; n < objects.size(); ++n)
    {
        rectangle(frameBgrOpenCv, cv::Rect(objects[n].x, objects[n].y,
            objects[n].width, objects[n].height),
            cv::Scalar(0, 0, 255), 1);
        putText(frameBgrOpenCv, std::to_string(objects[n].p),
            cv::Point(objects[n].x, objects[n].y),
            1, 1, cv::Scalar(0, 0, 255));
    }

    // Show video.
    cv::imshow("VIDEO", frameBgrOpenCv);
    // Wait ESC.
    if (cv::waitKey(1) == 27)
        return -1;
}

return 1;
}

```

