Gmd C++ library

v5.1.4

Table of contents

Overview

Gmd (Gauss Motion Detector) C++ library version **5.1.4** is designed for automatic detection of moving objects on videos. The library is implemented in C++ (C++17 standard) and utilizes the OpenMP library (2.5 standard) to facilitate parallel computation. It does not rely on any third-party code or include additional software libraries. The library is compatible **with any processors and operating systems** that support the C++ compiler (C++17 standard), provided the compiler has built-in support for the OpenMP (2.5 standard) parallel computing language. This library is suitable for various types of videos (daylight, SWIR, MWIR and LWIR), and it ensures accurate detection of small-size and low-contrast moving objects against complex backgrounds. Each instance of the Gmd C++ class performs frame-by-frame processing of a video data stream, processing each video frame independently. The library inherits its interface from the **ObjectDetector** class, offering flexible and customizable parameters. It seamlessly integrates into systems of varying complexity. The library is designed mainly for not moving cameras or for PTZ cameras when observing in a certain sector. The library is also used for object detection after the cameras have been moved by external command (drone detection systems). It also used to search for an object after the cameras are turned around by an external command in the direction of the object (C-UAS). The library compatible with low-power CPU.

Versions

Table 1 - Library versions.

Version	Release date	What's new
4.0.0	12.06.2020	First version.
5.0.0	11.09.2023	- Interface changed to ObjectDetector. - Motion mask calculation algorithm optimized. - Added frame buffer to provide detection on noisy video.
5.1.0	25.09.2023	- ObjectDetector interface updated. - Added decodeAndExecuteCommand(...) method.
5.1.1	30.10.2023	- getParams method updated.
5.1.2	06.11.2023	- Fixed processing time parameter calculation.
5.1.3	13.11.2023	- ObjectDetector class updated.
5.1.4	02.01.2024	- ObjectDetector class updated. - Demo application updated. - Documentation updated.

Library files

The library supplied by source code or compiled version. 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
    ObjectDetector -------------- files of ObjectDetector interface library
src ----------------------------- folder with library source code
    CMakeLists.txt -------------- CMake file
    Gmd.h ----------------------- main library header file
    GmdVersion.h ---------------- header file with library version
    GmdVersion.h.in ------------- file for CMake to generate version header
    Gmd.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
    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
```

When the library is delivered in compiled form, the user gets the following files:

```
CMakeLists.txt ------------------ main CMake file
README.md ----------------------- Documentation
3rdparty ------------------------ folder with 3rdparty libraries
    CMakeLists.txt -------------- CMake file for 3rdpary folder
    ObjectDetector -------------- files of ObjectDetector interface library
compiled ------------------------ Folder with CvTracker files.
    libGmd.a -------------------- Static library file for Linux OS.
    or libGmd.lib --------------- Static library file for Windows OS.
    Gmd.h ----------------------- Header file which includes Gmd class declaration.
    GmdVersion.h ---------------- Header file which includes version of the library.
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
    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
```

Gmd library depends on open source **ObjectDetector** (provides interface for object detector) which depends on open source **Frame** library (provides video frame structure and pixel formats description) and open source **ConfigReader** library (provides methods to work with JSON file and structures). Demo application depends on open source **SimpleFileDialog** library which provide file dialog function and also depends on **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 OpenMP library (version 2.5 and higher).
Supported OS	Compatible with any operating system that supports the C++ compiler (C++17 standard) and the OpenMP library (version 2.5 and higher).
Shape of detected objects	The library is able to detect moving objects of any 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 GRAY format for video processing. If the pixel format of the image is different from GRAY, the library pre-converts the pixel formats to GRAY.
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.
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 on the computing platform used. The processing time per video frame can be estimated with the demo application. It is possible to scale video frames to provide higher calculation speed.
Discreteness of computation of coordinates	The algorithm calculates the object bounding box for each detected object. The increment for calculation of position and parameters of the bounding box is 1 pixel.
Type of algorithm for detection of moving objects	A multi-hypothesis algorithm with the evaluation of movement trajectory and object parameters is used.

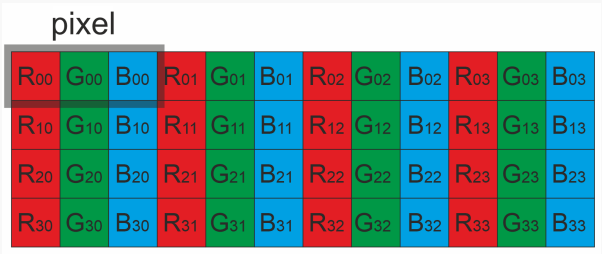
Parameter and feature	Description
Working conditions	Algorithms implemented in the library are designed to work on fixed cameras mainly. It is possible to work with slight camera movement depending on the background. It is also possible to operate while the camera is moving to search for moving objects against the sky. It is recommended to use a demo application to evaluate the quality of the algorithms in specific situations.

Supported pixel formats

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

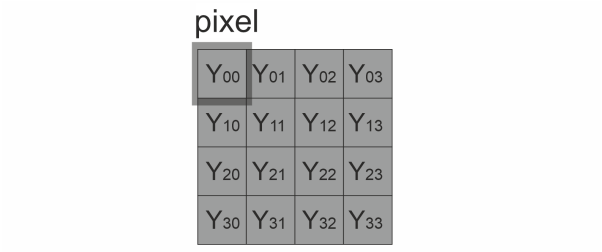
Table 3 - Bytes layout of supported RAW pixel formats. Example of 4x4 pixels image.



RGB24

BGR24

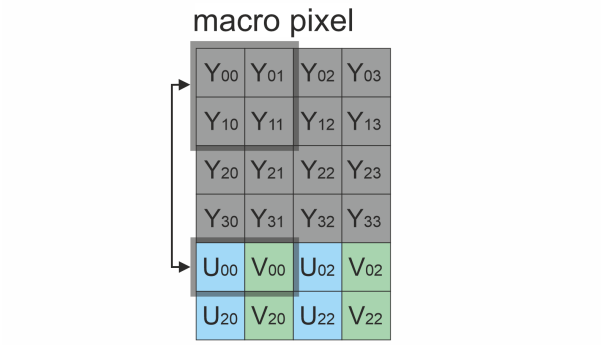
YUV24



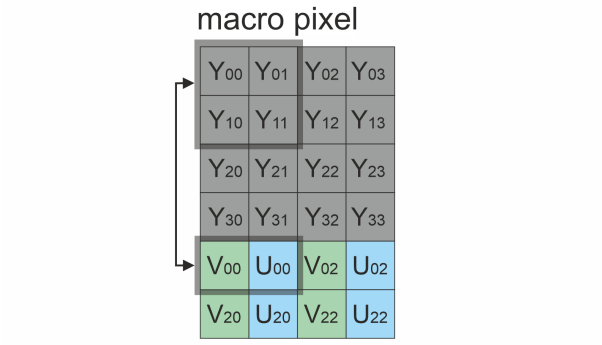
GRAY

YUYV

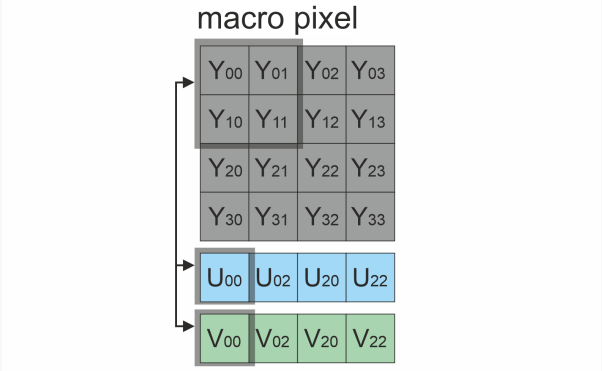
UYVY



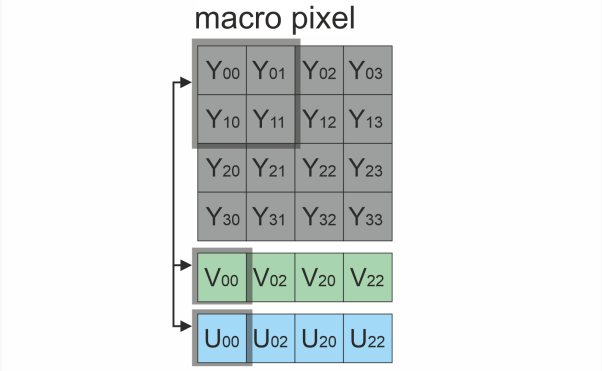
NV12



NV21



YU12



YV12

Library principles

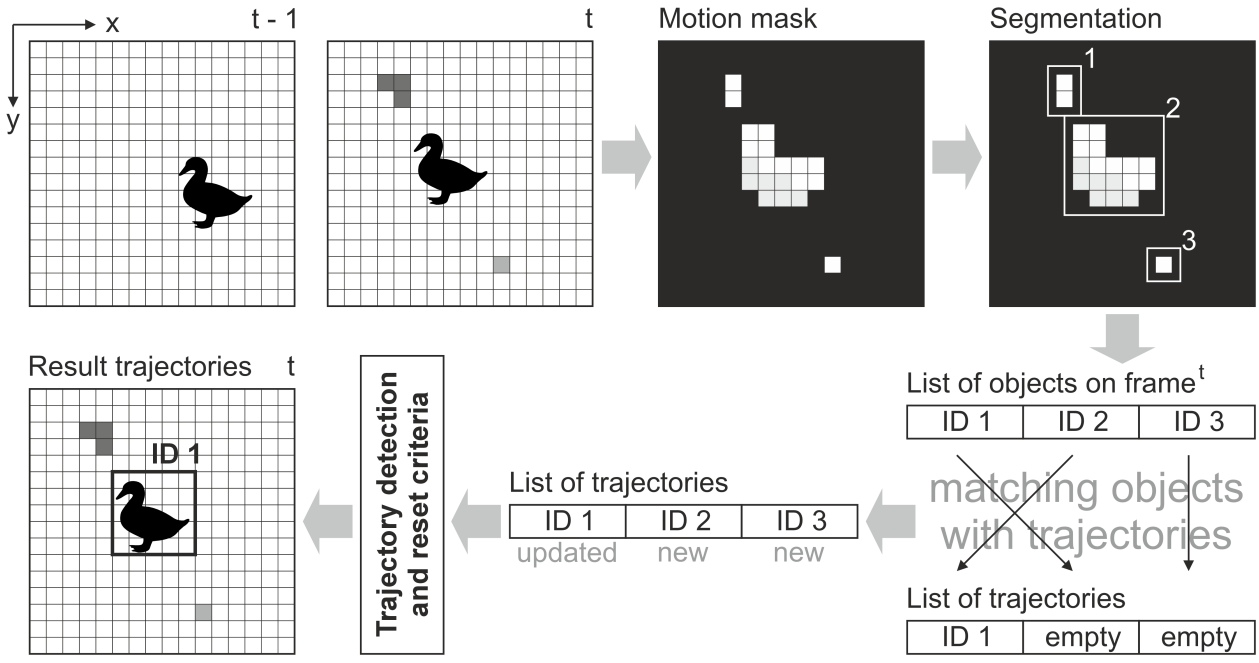
The object detection algorithm is implemented with multi-hypothesis support, incorporating movement trajectory evaluation and object parameter assessment. The algorithm involves the following sequential steps:

1. Acquiring the source video frame and converting it to GRAY format (grayscale).
2. Calculating the foreground mask (the most time-consuming operation, which can be multithreaded by setting the num thread parameter).
3. Generating a list of potential detected objects (motion mask segmentation).
4. Computing the correlation matrix of the probability that a detected object belongs to a track.
5. Determining if a track is sufficiently long to consider the object as detected.
6. The resulting objects for the current frame can be retrieved using the getObjects() method.

The library is available as either source code or a compiled application. 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 Gmd C++ class. If you need multiple parallel cameras processing you have to create multiple Gmd C++ class instances.
3. If necessary, modify the default library parameters using the setParam(...) method.
4. Create Frame class object for input frame and a vector of Objects.
5. Call the detect(...) method to identify objects.
6. Fill the created vector of Objects using the getObjects() method.

Motion detection principles:



Gmd class description

Gmd class declaration

Gmd.h file contains **Gmd** class declaration. **Gmd** class inherits interface from **ObjectDetector** interface class. Class declaration:

```
class Gmd : 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.
    bool decodeAndExecuteCommand(uint8_t* data, int size) override;

    /// This method retrieves the motion detection binary mask.
    bool getMotionMask(cr::video::Frame& mask);
}
```

getVersion method

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

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

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

```
cout << "Gmd version: " << Gmd::getVersion() << endl;
```

Console output:

```
Gmd version: 5.1.4
```

initObjectDetector method

initObjectDetector(...) method initializes motion detector. 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: frameBufferSize (Default value 1) minObjectWidth (Default value 2) maxObjectWidth (Default value 128) minObjectHeight (Default value 2) maxObjectHeight (Default value 128) sensitivity (Default value 10) scaleFactor (Default value 1) numThreads (Default value 1) minXSpeed (Default value 0.1) maxXSpeed (Default value 15) minYSpeed (Default value 0.1) maxYSpeed (Default value 15) xDetectionCriteria (Default value 10) yDetectionCriteria (Default value 10) resetCriteria (Default value 10) If particular parameter out of valid range the library will set default values automatically.

Returns: always returns TRUE.

setParam method

setParam(...) method designed to set new Gmd 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(ObjectDetectorParam id, float value) override;
```

Parameter	Description
id	Parameter ID according to **ObjectDetectorParam** enum (defined by **ObjectDetector** interface class). The library supports not all parameters from **ObjectDetector** interface.
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. **getParam(...)** is thread-safe method. This means that the **getParam(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
id	Parameter ID according to **ObjectDetectorParam** enum (defined by **ObjectDetector** interface class). The library supports not all parameters from **ObjectDetector** interface.

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

getParams method

getParams(...) method designed to obtain object detector params structures as well a list of detected objects (included in **ObjectDetectorParams** class). **getParams(...)** is thread-safe method. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

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

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

getObjects method

getObjects() method returns list of detected objects. User can obtain object 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 detected object the list will be empty.

executeCommand method

executeCommand(...) method designed to execute object detector command. **executeCommand(...)** is thread-safe method. This means that the **executeCommand(...)** method can be safely called from any thread. Method declaration:

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

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

Returns: TRUE is 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). The library uses the GRAY format for video processing. If the pixel format of the image is different from GRAY, the library pre-converts the pixel formats to GRAY.

Returns: TRUE if the video frame was processed FALSE if not. If object detector disabled (see **ObjectDetectorParam** enum description) the method should return 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 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	Detection mask is see **Frame** object with GRAY pixel format. Detector omits image segments, where detection mask pixel values equal 0. Mask can have any resolution. If resolution of mask (width and height) 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).

getMotionMask method

getMotionMask(...) method retrieves the motion detection binary mask, that is utilized by the detector to identify objects in the video stream. Method not included in **ObjectDetector** interface. Method declaration:

```
bool getMotionMask(cr::video::Frame& mask);
```

Parameter	Description
mask	Image of motion mask. Must have GRAY (preferable), NV12, NV21, YV12 or YU12 pixel format. If Frame object not initialized the library will initialize it.

Returns: TRUE if the mask image is filled or FALSE if not (not valid pixel format).

decodeAndExecuteCommand method

decodeAndExecuteCommand(...) method decodes and executes command on object detector side. **decodeAndExecuteCommand(...)** is thread-safe method. This means that the **decodeAndExecuteCommand(...)** method can be safely called from any thread. 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 designed to encode command to change any parameter for remote object detector. 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 (11 bytes):

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	Major version	Major version of ObjectDetector class.
2	Minor version	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.
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 designed to encode command for remote object detector. 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 >= 7.
size	Size of encoded data. Will be 7 bytes.
id	Command ID according to ObjectDetectorCommand enum.

COMMAND format (7 bytes):

Byte	Value	Description
0	0x00	COMMAND header value.
1	Major version	Major version of ObjectDetector class.
2	Minor version	Minor version of ObjectDetector class.
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[7];
// Size of encoded data.
int size = 0;
// Encode command.
ObjectDetector::encodeCommand(data, size, ObjectDetectorCommand::RESET);
```

decodeCommand method of ObjectDetector class

decodeCommand(...) static method 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 - Gmd class params description (from **ObjectDetector** interface class). Some params may be unsupported by Gmd class.

Parameter	Access	Description
LOG_MODE	read / write	**Not used.** Can have any values.
FRAME_BUFFER_SIZE	read / write	Frame buffer size. Valid values from 1 to 128. If the frame buffer size is 1, the library converts the frame format to GRAY and processes it. If the buffer size is greater than 1, the library will calculate the average value of brightness (converted to GRAY format) of each pixel of the frame over several frames (FRAME_BUFFER_SIZE) and use it as input data. In this way, the original video is time smoothed. This allows you to detect objects in the video with high noise.
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.

Parameter	Access	Description
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	Minimum object's horizontal speed to be detected, pixels/frame. Valid values from 0 to 256. Must be < MAX_X_SPEED. To be detected object's horizontal speed must be >= MIN_X_SPEED. Recommended value is 0.1.
MAX_X_SPEED	read / write	Maximum object's horizontal speed to be detected, pixels/frame. Valid values from 0 to 256. Must be > MIN_X_SPEED. To be detected object's horizontal speed must be <= MAX_X_SPEED. Recommended value is 15.
MIN_Y_SPEED	read / write	Minimum object's vertical speed to be detected, pixels/frame. Valid values from 0 to 256. Must be < MAX_Y_SPEED. To be detected object's vertical speed must be >= MIN_Y_SPEED. Recommended value is 0.1.
MAX_Y_SPEED	read / write	Maximum object's vertical speed to be detected, pixels/frame. Valid values from 0 to 256. Must be > MIN_Y_SPEED. To be detected object's vertical speed must be <= MAX_Y_SPEED. Recommended value is 15.
MIN_DETECTION_PROPABILITY	read / write	**Not used.** Can have any values.
X_DETECTION_CRITERIA	read / write	Horizontal track detection criteria, frames. Shows how many frames the objects must move in any(+/-) horizontal direction to be detected.
Y_DETECTION_CRITERIA	read / write	Vertical track detection criteria, frames. Shows how many frames the objects must move in any(+/-) vertical direction to be detected.
RESET_CRITERIA	read / write	Track reset criteria, frames. Shows how many frames the objects should be not detected to be excluded from results.

Parameter	Access	Description
SENSITIVITY	read / write	Detection sensitivity. For Gmd library this parameters means pixel brightness deviation threshold from 0 to 255 for calculation motion mask. The first processing step is the computation of the binary motion mask. The algorithm makes a decision about the presence of motion in a particular pixel also on the basis of changes in pixel brightness. The brightness threshold is determined by the SENSITIVITY parameter. The lower this parameter is, the less contrasty objects will be detected, but there will be more influence of video noise. The larger this parameter is, the more contrast objects must have in order to be detected. The normal value for thermal cameras is **15**. The normal value for daylight cameras is **10**.
SCALE_FACTOR	read / write	Frame scaling factor for processing purposes. Reduce the image size by scaleFactor times horizontally and vertically for faster processing. Allows to increase calculation speed but slightly reduces sensitivity.
NUM_THREADS	read / write	Num threads. Number of threads for parallel computing. Multiple threads are used to compute the binary motion mask.
PROCESSING_TIME_MKS	read only	Processing time for last frame, mks. The library independently calculates the processing time for each frame of video.
TYPE	read / write	**Not used.** Can have any values.
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 values.
CUSTOM_2	read / write	**Not used.** Can have any values.
CUSTOM_3	read / write	**Not used.** Can have any values.

Object structure

Object structure used to describe detected object. Object structure declared in **ObjectDetector.h** file and also included in **ObjectDetectoParams** structure. 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. Unique for particular object. The library assigns a unique ID to the frame of a new detected object and does not change it from frame to frame. If the object disappears and reappears, the library can assign a new ID. This is necessary for control algorithms to distinguish different objects from frame to frame.
frameId	int	Frame ID. Will be the same as frame ID of processed video frame.
type	int	Object type. Depends on implementation. For example detection neural networks can detect multiple type of objects.
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.

Field	Type	Description
vX	float	Horizontal component of object velocity, +-pixels/frame. Object detector estimates object velocity on video frames.
vY	float	Vertical component of object velocity, +-pixels/frame. Object detector estimates object velocity on video frames.
p	float	**Not used.** Will have value 1.

ObjectDetectorParams class description

ObjectDetectorParams class declaration

ObjectDetectorParams class used for object detector initialization (**initObjectDetector(...)** method) or to get all actual params (**getParams()** method). Also **ObjectDetectorParams** provide structure to write/read params from JSON files (**JSON_READABLE** macro, see ConfigReader class description) and provide methos 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);

    /// operator =
    ObjectDetectorParams& operator= (const ObjectDetectorParams& src);

    /// Encode params.
    bool encode(uint8_t* data, int bufferSize, int& size,
                ObjectDetectorParamsMask* mask = nullptr);

    /// Decode params.
```

```
    bool decode(uint8_t* data, int dataSize);
};
```

Table 7 - ObjectDetectorParams class fields description.

Field	Type	Description
initString	string	**Not used.** Can have any values.
logMode	int	Logging mode. Valid values: 0 - Disable, 1 - Only file, 2 - Only terminal, 3 - File and terminal.
frameBufferSize	int	Frame buffer size. Valid values from 1 to 128. If the frame buffer size is 1, the library converts the frame format to GRAY and processes it. If the buffer size is greater than 1, the library will calculate the average value of brightness (converted to GRAY format) of each pixel of the frame over several frames (FRAME_BUFFER_SIZE) and use it as input data. In this way, the original video is time smoothed. This allows you to detect objects in the video with high noise.
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.
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	Minimum object's horizontal speed to be detected, pixels/frame. Valid values from 0 to 256. Must be < maxXSpeed. To be detected object's horizontal speed must be >= minXSpeed. Recommended value is 0.1.
maxXSpeed	float	Maximum object's horizontal speed to be detected, pixels/frame. Valid values from 0 to 256. Must be > minXSpeed. To be detected object's horizontal speed must be <= maxXSpeed. Recommended value is 15.

Field	Type	Description
minYSpeed	float	Minimum object's vertical speed to be detected, pixels/frame. Valid values from 0 to 256. Must be < maxYSpeed. To be detected object's vertical speed must be >= minYSpeed. Recommended value is 0.1.
maxYSpeed	float	Maximum object's vertical speed to be detected, pixels/frame. Valid values from 0 to 256. Must be > minYSpeed. To be detected object's vertical speed must be <= maxYSpeed. Recommended value is 15.
minDetectionProbability	float	**Not used.** Can have any values.
xDetectionCriteria	int	Horizontal track detection criteria, frames. Shows how many frames the objects must move in any(+/-) horizontal direction to be detected.
yDetectionCriteria	int	Vertical track detection criteria, frames. Shows how many frames the objects must move in any(+/-) vertical direction to be detected.
resetCriteria	int	Track reset criteria, frames. Shows how many frames the objects should be not detected to be excluded from results.
sensitivity	float	Detection sensitivity. For Gmd library this parameters means pixel brightness deviation threshold from 0 to 255 for calculation motion mask. The first processing step is the computation of the binary motion mask. The algorithm makes a decision about the presence of motion in a particular pixel also on the basis of changes in pixel brightness. The brightness threshold is determined by the SENSITIVITY parameter. The lower this parameter is, the less contrasty objects will be detected, but there will be more influence of video noise. The larger this parameter is, the more contrast objects must have in order to be detected. The normal value for thermal cameras is **15**. The normal value for daylight cameras is **10**.
scaleFactor	int	Frame scaling factor for processing purposes. Reduce the image size by scaleFactor times horizontally and vertically for faster processing. Allows to increase calculation speed but slightly reduces sensitivity.
numThreads	int	Num threads. Number of threads for parallel computing. Multiple threads are used to compute the binary motion mask.
processingTimeMks	int	Processing time for last frame, mks. The library independently calculates the processing time for each frame of video.
type	int	**Not used.** Can have any values.

Field	Type	Description
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 values.
custom2	float	**Not used.** Can have any values.
custom3	float	**Not used.** Can have any values.
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:

```
bool encode(uint8_t* data, int bufferSize, 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 ObjectDetector.h 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 ObjectDetectorParamsMask structure.

Returns: TRUE is params serialized or FALSE if not.

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, 1024, 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, 1024, 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, int dataSize);
```

Parameter	Value
data	Pointer to encode data buffer.
dataSize	Data size.

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, 1024, size);
cout << "Encoded data size: " << size << " bytes" << endl;

// Decode data.
ObjectDetectorParams out;
if (!out.decode(data, size))
{
    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

Typical commands to build **Gmd** library:

```
cd Gmd
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want to connect **Gmd** 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 **Gmd** as git submodule by commands (or just copy repository files):

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

In your repository folder, a new **3rdparty/Gmd** folder will be created, which contains files from **Gmd** repository along with its subrepository **ObjectDetector** and its subrepositories **Frame** and **ConfigReader**. If you don't have access to GitHub repository, copy **Gmd** 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
    Gmd
```

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(${PARENT}_SUBMODULE_GMD                                 ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_GMD)
    SET(${PARENT}_GMD                                       ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_GMD_TEST                                  OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_GMD_DEMO_APP                              OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_GMD_EXAMPLE                               OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

Done!

Simple example

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

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

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

    // Create detector and set params.
    cr::detector::Gmd detector;
    detector.setParam(cr::detector::ObjectDetectorParam::MIN_OBJECT_WIDTH, 4);
    detector.setParam(cr::detector::ObjectDetectorParam::MAX_OBJECT_WIDTH, 96);
    detector.setParam(cr::detector::ObjectDetectorParam::MIN_OBJECT_HEIGHT, 4);
    detector.setParam(cr::detector::ObjectDetectorParam::MAX_OBJECT_HEIGHT, 96);
    detector.setParam(cr::detector::ObjectDetectorParam::SENSITIVITY, 10);

    // Create frames.
    cv::Mat frameBgrOpenCv;

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

        // Create Frame object.
        cr::video::Frame bgrFrame;
        bgrFrame.width = frameBgrOpenCv.size().width;
        bgrFrame.height = frameBgrOpenCv.size().height;
        bgrFrame.size = bgrFrame.width * bgrFrame.height * 3;
        bgrFrame.data = frameBgrOpenCv.data;
        bgrFrame.fourcc = cr::video::Fourcc::BGR24;

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

        // 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);
        }

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

        // Wait ESC.
        if (cv::waitKey(1) == 27)
            return -1;
    }

    return 1;
}
```