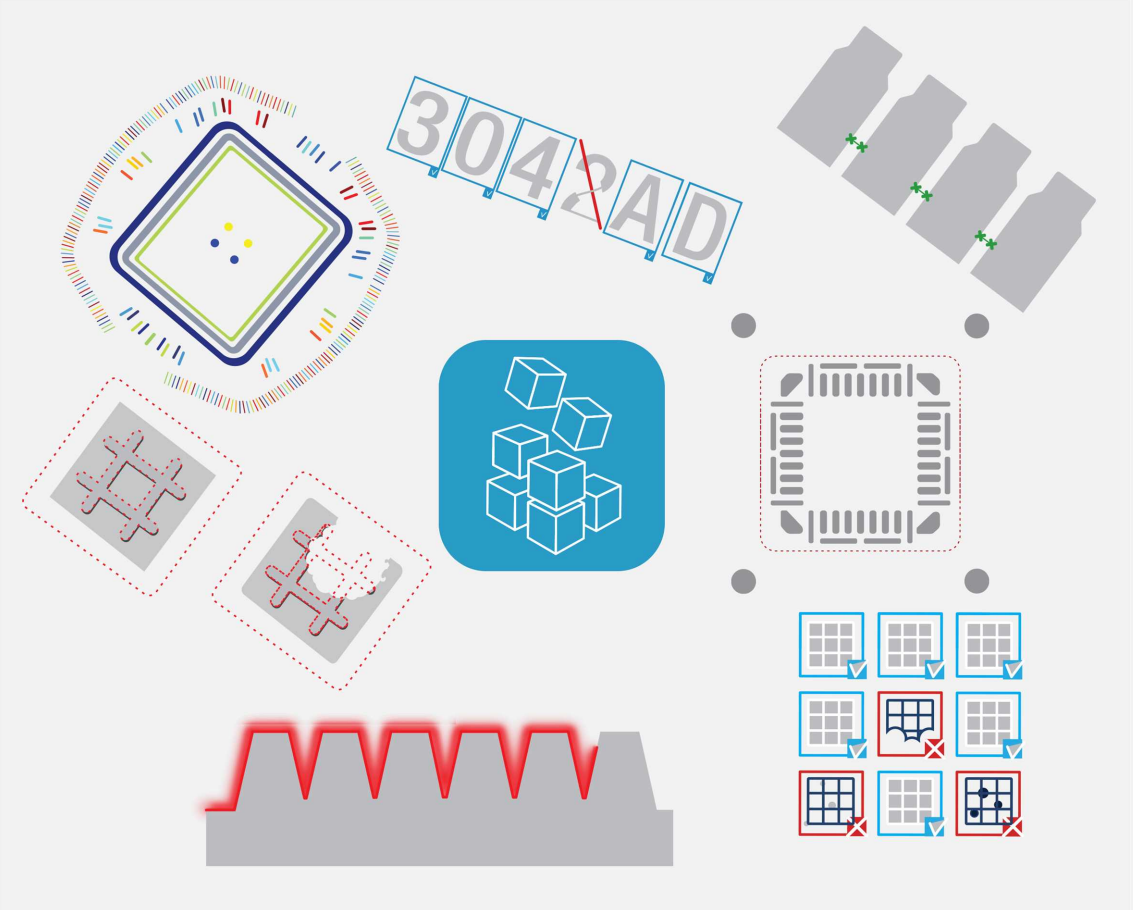
euresys
Empowering Computer Vision

USER GUIDE

Open eVision

Easy3D Compatibility with LMI Gocator 3D Sensors



Terms of Use

EURESYS s.a. shall retain all property rights, title and interest of the documentation of the hardware and the software, and of the trademarks of EURESYS s.a.

All the names of companies and products mentioned in the documentation may be the trademarks of their respective owners.

The licensing, use, leasing, loaning, translation, reproduction, copying or modification of the hardware or the software, brands or documentation of EURESYS s.a. contained in this book, is not allowed without prior notice.

EURESYS s.a. may modify the product specification or change the information given in this documentation at any time, at its discretion, and without prior notice.

EURESYS s.a. shall not be liable for any loss of or damage to revenues, profits, goodwill, data, information systems or other special, incidental, indirect, consequential or punitive damages of any kind arising in connection with the use of the hardware or the software of EURESYS s.a. or resulting of omissions or errors in this documentation.

This documentation is provided with Open eVision 2.12.1 (doc build 1132).
www.euresys.com

Easy3D Compatibility with LMI Gocator 3D Sensors

Introduction

LMI technologies Gocator sensors are integrated 3D laser profilers. They perform high speed and high resolution 3D scanning using laser line triangulation principle. LMI provides the Gocator SDK for integration of these devices in third party applications.

The specifications are available on the manufacturer website:
`https://lmi3d.com/products/gocator-3D-smart-sensors`

- This document explains how to use the 3D data coming from these sensors with **Open eVision** 3D libraries and tools.
- A sample application distributed with source code demonstrates that integration.

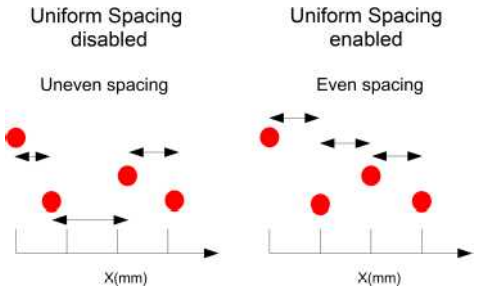
Resources

This document and the sample applications are based on the following resources:

- Gocator 2430 device
- Gocator Software Development Kit 5.3.22
- Open eVision 2.12
- Microsoft Visual Studio 2017

Features

- The Gocator SDK provides 2 types of 3D surface data:
  - *Point cloud data*: the original 3D data, unevenly spaced on X axis.
    - The positions are defined by (X, Y, Z) coordinates.
  - *Uniform sampling data*: resampled data along the X axis.
    - You can configure the spacing in the Sensor panel on the Scan page.
    - The positions are reported as a single Z coordinate per pixel.



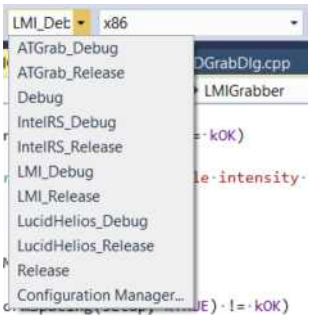
- A specific value is used for “undefined” position. These 3D positions are already sensor calibrated and expressed in metric coordinate system.
- These 2 formats match the Open eVision **Easy3D** classes `EPointCloud` and `EZMap`, so the conversion is straightforward.
  - Original 3D data from Gocator SDK are 16-bit numbers. Convert them to floating point values before filling the `EPointCloud`.
  - **Easy3D** ZMap class natively supports 16-bit data.
- Do not use the Open eVision `EDepthMap` container. It is designed for uncalibrated depth or range data and the Gocator device does not provide that format.

> **TIP**
> The **Easy3DGrab** sample application implements a simple acquisition and display the pipeline for LMI Gocator sensors.

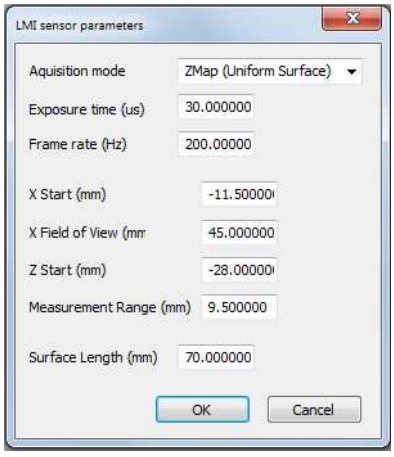
Easy3DGrab sample application

- **Easy3DGrab** is distributed with C++ source code as an Open eVision additional resource.
  - It features the acquisition of **LMI** Gocator 3D data and the conversion to point clouds and Zmaps.
  - You can save these representations.
  - Click on the Grab button to acquire a new image.
  - Open the Sensor Properties dialog to access some of the device parameters.
  - The Object extraction function is exposed but you can use it only with the Easy3DObject license.

- The **Easy3DGrabVisual Studio** project includes different configurations for different 3D sensors: Automation technologies, Intel RealSense, LMI Gocator, Lucid Helios… To build the application, you usually need to install the corresponding SDK from the sensor manufacturer.



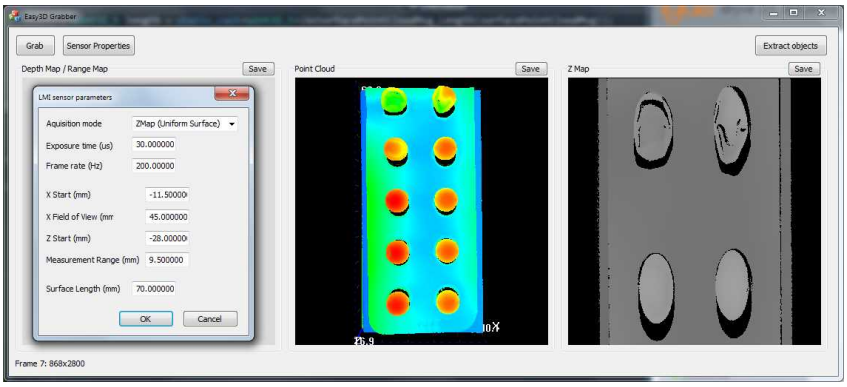
- A subset of parameters is exposed in the sensor parameters dialog:
  - The Acquisition mode to set the 3D data as Point cloud (Uneven Spacing) or ZMap (Uniform Surface).
  - The Exposure time (µs) set in microseconds.
  - The Frame rate (Hz) of the sensor.
    The **Easy3DGrab** sample uses only the time as trigger source and not the encoder. But you can configure the sensor otherwise.
  - The X Start (mm), X field of View (mm), Z Start (mm) and Measurement Range (mm) define the active area.
  - the Surface Length (mm) set the scan length along the Y axis.
    The effective number of captured lines depends on this parameter, the frame rate and the travel speed (defined in the Motion and Alignment tab of the web interface).
- The default values for these parameters are in the `LMIGrabber.h` source code file.



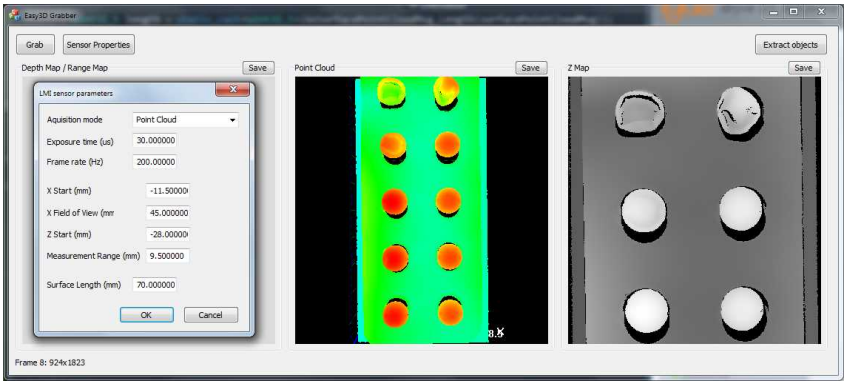
TIP
You can edit all the other parameters in the sensor Web interface.

Examples

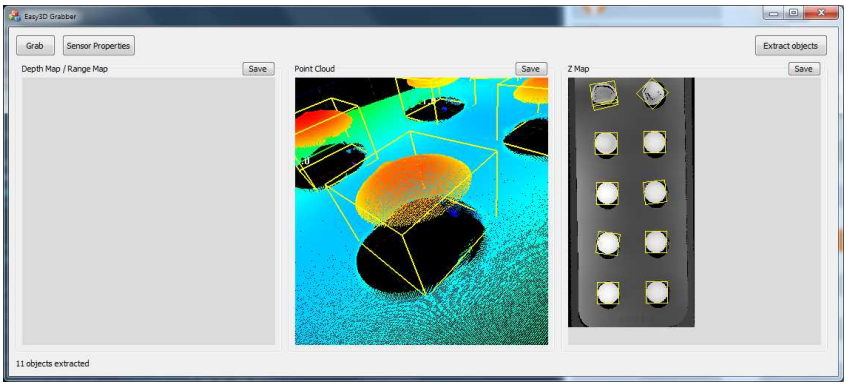
- Acquisition of a ZMap (uniform surface configuration).
  - The left dialog shows the current sensor properties.
  - The point cloud is generated from the ZMap using the `EZMapToPointCloudConverter` class.
  - The ZMap is displayed as an image so it does not reflect the correct metric scale.



- Acquisition of a Point Cloud (non uniform surface configuration).
  - The ZMap is generated from the point cloud using the `EPointCloudToZMapConverter` class.



- Extraction of the 3D object with the **Easy3DObject** library.
  - The segmented objects are highlighted in the 3D and the 2D views.



C++ code sample to convert GoDataMsg to Easy3D objects

```
GoDataMsg dataObj = GoDataSet_At(dataset, i);

switch (GoDataMsg_Type(dataObj))
{
case GO_DATA_MESSAGE_TYPE_SURFACE_POINT_CLOUD:
{
  GoSurfacePointCloudMsg surfacePointCloudMsg = dataObj;

  float XResolution = static_cast<float>(NM_TO_MM(GoSurfacePointCloudMsg_
XResolution(surfacePointCloudMsg)));
  float YResolution = static_cast<float>(NM_TO_MM(GoSurfacePointCloudMsg_
YResolution(surfacePointCloudMsg)));
  float ZResolution = static_cast<float>(NM_TO_MM(GoSurfacePointCloudMsg_
ZResolution(surfacePointCloudMsg)));
  float XOffset = static_cast<float>(UM_TO_MM(GoSurfacePointCloudMsg_XOffset
(surfacePointCloudMsg)));
  float YOffset = static_cast<float>(UM_TO_MM(GoSurfacePointCloudMsg_YOffset
(surfacePointCloudMsg)));
  float ZOffset = static_cast<float>(UM_TO_MM(GoSurfacePointCloudMsg_ZOffset
(surfacePointCloudMsg)));

  uint32_t width = static_cast<uint32_t>(GoSurfacePointCloudMsg_Width
(surfacePointCloudMsg));
  uint32_t length = static_cast<uint32_t>(GoSurfacePointCloudMsg_Length
(surfacePointCloudMsg));

  std::vector<Easy3D::E3DPoint> pts;
  pts.reserve(width * length);

  Easy3D::E3DPoint p;
  for (uint32_t y = 0; y < length; y++)
  {
    kPoint3d16s *data = GoSurfacePointCloudMsg_RowAt(surfacePointCloudMsg, y);

    for (uint32_t x = 0; x < width; x++)
    {
      if (data[x].z != INVALID_RANGE_16BIT)
      {
        p.X = XOffset + XResolution * data[x].x;
        p.Y = YOffset + YResolution * data[x].y;
        p.Z = ZOffset + ZResolution * data[x].z;
        pts.push_back(p);
      }
    }
  }

  point_cloud = new Easy3D::EPointCloud();
  point_cloud->AddPoints(pts);
}
break;

case GO_DATA_MESSAGE_TYPE_UNIFORM_SURFACE:
{
  GoSurfaceMsg surfaceMsg = dataObj;

  uint32_t width = static_cast<uint32_t>(GoSurfaceMsg_Width(surfaceMsg));
  uint32_t length = static_cast<uint32_t>(GoSurfaceMsg_Length(surfaceMsg));

  float XResolution = static_cast<float>(NM_TO_MM(GoSurfaceMsg_XResolution
(surfaceMsg)));
```

```
  float YResolution = static_cast<float>(NM_TO_MM(GoSurfaceMsg_YResolution
(surfaceMsg)));
  float ZResolution = static_cast<float>(NM_TO_MM(GoSurfaceMsg_ZResolution
(surfaceMsg)));

  zmap = new Easy3D::EZMap16(width, length);
  zmap->SetResolution(XResolution, YResolution, ZResolution);
  uint16_t undef = zmap->GetUndefinedValue().Value;

  // Copy buffer to EZMap16
  for (uint32_t y = 0; y < length; ++y)
  {
    k16s *data = GoSurfaceMsg_RowAt(surfaceMsg, y);
    uint16_t *dst = (uint16_t*)zmap->GetCheckedBufferPtr(0, y);
    for (uint32_t x = 0; x < width; ++x, ++dst)
    {
      // use only valid data
      if (data[x] != INVALID_RANGE_16BIT)
      {
        *dst = data[x] - SHRT_MIN;
      }
      else
      {
        *dst = undef;
      }
    }
  }
}
break;
}
```