Python Gocator Integration Guide

Running the Example 3

Cases of Interest 4

- Stamp 5
- Measurements 6
- Uniform Surfaces 7
- Intensity Image 9

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com

HEADQUARTERS
9200 Glenlyon Parkway
Burnaby, BC V5J 5J8
Canada

Overview

Python is a general purpose programming language that is commonly used in machine learning applications. This guide is intended to support users who are interested in getting data from a Gocator and processing it using Python. The Gocator SDK, or GoSDK, is a DLL which uses C functions to interact with Gocator sensors via the Gocator protocol. This guide will use the ctypes library to interact with the GoSDK DLL to receive data and save it to a file.

Getting Started

You must have the following installed on Linux or Windows:

- Python 3.7 or above
- Numpy (pip install numpy)
- Pillow (pip install Pillow)
- OpenCV (pip install opencv-python)
- GoSDK package from LMI3D.com (the latest version will do)

Windows Requirements

Ensure the GoSDK file has been unzipped. The first step to loading GoSDK.dll is to create an environmental variable pointing to the GoSDK root directory. The root directory is named GO_SDK .

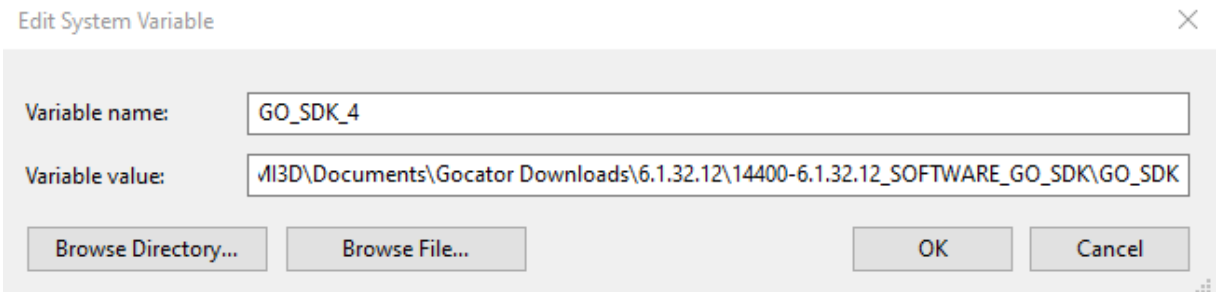

Figure 1. System variable configuration on Windows systems.

```
### Load Api
# Please define your System Environment Variable as GO_SDK_4. It should reference the root directory of the SDK package.
# If you would like to compile a 64-bit version of this project. Please change the win32 part of the dll paths to win64 for a Windows64-bit OS
SdkPath = os.environ['GO_SDK_4']
kApi = ctypes.windll.LoadLibrary(os.path.join(SdkPath, 'bin', 'win64', 'kApi.dll'))
GoSdk = ctypes.windll.LoadLibrary(os.path.join(SdkPath, 'bin', 'win64', 'GoSdk.dll'))
```

Figure 2. `ctypes` loading call for `kApi.dll` and `GoSdk.dll` for Windows.

Linux Requirements

Ensure the GoSDK file has been unzipped. The Linux .so files need to be built. You must ensure that `make` and `build-essential` have been installed:

```
apt install make
apt install build-essential
```

Next, run the `makefile` for the architecture you're using. For desktop PC's, this will typically be X64 and located in `GO_SDK/Gocator`. Then run the following to make the required `.so` files:

```
make -f GoSdk-Linux_X64.mk
```

The output directory for the .so files are located in `GO_SDK/lib/linux_x64d`, for X64 make files.

Now that GoSDK is built, we can point a variable to the GoSDK root directory:

```
GO_SDK_4='/home/Downloads/GO_SDK' (or wherever your GO_SDK root
directory is)
export GO_SDK_4
```

To verify, run `printenv` and ensure `GO_SDK_4` is listed.

The system should now be able to load the required libraries.

```
#Linux
kApi = ctypes.cdll.LoadLibrary(os.path.join(SdkPath, 'lib', 'linux_x64d', 'libkApi.so'))
GoSdk = ctypes.cdll.LoadLibrary(os.path.join(SdkPath, 'lib', 'linux_x64d', 'libGoSdk.so'))
```

Figure 3. `ctypes` loading call for Linux systems.

Running the Example

The example should now be ready to run, but first a sensor needs to be on the network. If you don't have a sensor, open an emulator scenario (Windows only). If you're running a Linux docker container, you can run an emulator scenario open to the network via command line in Windows. Example:

`.\GoEmulator.exe /ip 10.0.0.79` (whatever the local IP address is)

Run the example:

```
python3 .\Gocator_Python_Example.py
```

If there is no sensor connected, an access violation error will occur. This will be the case if the required ports are closed on your system. Ensure the ports are open and try again. If you don't see an error message, but nothing else, you're likely already connected to the Gocator and are waiting for new data to be acquired.

Use Case Scenarios

GoSdk is a protocol interpreter and message buffer for Gocator systems. The sensor will send messages containing frame information and data. For example, a message can contain measurements, surface data, and profile data. Each message is buffered and sent in order through GoSdk. Messages are prepended with a stamp that gives timing and frame information about the message payload.

The other items in a message are measurement, surface and profile information.

These items are sent and looped over the following snippet:

```
109     ### Receive Measurement Data
110     while(1):
111         #Place this in a while loop for continuous calling. Ondata callback isn't available
112         if (GoSdk.GoSystem_ReceiveData(system, byref(dataset), RECEIVE_TIMEOUT) == kOK):
113             print("Data message received:")
114             print("Dataset count: %u" % GoSdk.GoDataSet_Count(dataset))
115 
116             ## loop through all items in result message
117             for i in range(GoSdk.GoDataSet_Count(dataset)):
118                 k_object_address = GoSdk.GoDataSet_At(dataset, i)
119                 dataObj = GoDataMsg(k_object_address)
```

Figure 4. Line 112 checks for a system message. If one exists the count is pulled and a `for` loop iterates through the items in the message.

Stamp

A `GoStamp` is sent regardless of the selected output types. It will even be sent if nothing is selected on the **Output** tab in the Gocator web interface. The class is defined via the GoStamp struct in GoSdk.

```
class GoStampData(Structure):
    _fields_ = [("frameIndex", c_uint64), ("timestamp",c_uint64), ("encoder", c_int64), ("encoderAtZ", c_int64), ("status", c_uint64), ("id", c_uint32)]
```

```
GoSdk.GoStampMsg_At.restype = ctypes.POINTER(GoStampData)
```

Figure 5. `GoStampData` class and pointer definition.

```
123                 ## Retrieve stamp message
124                 if GoSdk.GoDataMsg_Type(dataObj) == GO_DATA_MESSAGE_TYPE_STAMP:
125                     stampMsg = dataObj
126                     msgCount = GoSdk.GoStampMsg_Count(stampMsg)
127
128                     for k in range(msgCount):
129                         stampDataPtr = GoSdk.GoStampMsg_At(stampMsg,k)
130                         stampData = stampDataPtr.contents
131                         print("frame index: ", stampData.frameIndex)
132                         print("time stamp: ", stampData.timestamp)
133                         print("encoder: ", stampData.encoder)
134                         print("sensor ID: ", stampData.id)
```

Figure 6. GoStamp message case. The GoStamp pointer is cast to GoStampData, which is then printed based on the class definition.

Measurements

GoMeasurement data is set up in much the same way as GoStamp data in ctypes.

```
class GoMeasurementData(Structure):
    _fields_ = [("numericVal", c_double), ("decision", c_uint8), ("decisionCode", c_uint8)]
```

```
GoSdk.GoMeasurementMsg_At.restype = ctypes.POINTER(GoMeasurementData)
```

Figure 7. GoMeasurement message class and pointer definition.

A measurement message contains measurement values, decision data (pass/fail), and decision code (fail, invalid, invalid anchor).

```
136             # Retrieve measurement data messages
137             if GoSdk.GoDataMsg_Type(dataObj) == GO_DATA_MESSAGE_TYPE_MEASUREMENT:
138                 measurementMsg = dataObj
139                 msgCount = GoSdk.GoMeasurementMsg_Count(measurementMsg)
140                 print("Measurement Message batch count: %d" % msgCount);
141
142                 for k in range(GoSdk.GoMeasurementMsg_Count(measurementMsg)):
143                     measurementDataPtr = (GoSdk.GoMeasurementMsg_At(measurementMsg, k))
144                     measurementData = measurementDataPtr.contents #(measurementDataPtr, POINTER(GoMeasurementData)).contents
145                     measurementID = (GoSdk.GoMeasurementMsg_Id(measurementMsg))
146                     print("Measurement ID: ", measurementID)
147                     print("Measurement Value: ", measurementData.numericVal)
148                     print("Measurment Decision: " + str(measurementData.decision))
```

Figure 8. GoMeasurement message class implementation.

When operating properly, the output should look like figure 9 for one measurement output.

```
Data message received:
Dataset count: 2
frame index:  1
time stamp:  6580943237
encoder:  0
sensor ID:  55349
Measurement Message batch count: 1
Measurement ID:  42
Measurement Value:  -54.784
Measurment Decision: 0
```

Figure 9. Python output values for one valid measurement.

Uniform Surfaces

Uniform Surfaces are height maps with a defined X and Y resolution, and a unique Z value for each point. It's essentially a 2D image where each value represents a height value instead of color or intensity. Data contained in the message is as follows:

- X, Y, Z resolution
- X, Y, Z offset
- Width
- Length
- 1D Z array

With the above data, the height map can be reconstructed as seen on the sensor.

```
#resolutions and offsets (cast to mm)
XResolution = float((GoSdk.GoUniformSurfaceMsg_XResolution(surfaceMsg)))/1000000.0
YResolution = float((GoSdk.GoUniformSurfaceMsg_YResolution(surfaceMsg)))/1000000.0
ZResolution = float((GoSdk.GoUniformSurfaceMsg_ZResolution(surfaceMsg)))/1000000.0
XOffset = float((GoSdk.GoUniformSurfaceMsg_XOffset(surfaceMsg)))/1000.0
YOffset = float((GoSdk.GoUniformSurfaceMsg_YOffset(surfaceMsg)))/1000.0
ZOffset = float((GoSdk.GoUniformSurfaceMsg_ZOffset(surfaceMsg)))/1000.0
width = GoSdk.GoUniformSurfaceMsg_Width(surfaceMsg)
length = GoSdk.GoUniformSurfaceMsg_Length(surfaceMsg)
size = width * length
```

Figure 10. Resolutions and offsets are cast to mm, total size is calculated.

```
GoSdk.GoUniformSurfaceMsg_RowAt.restype = ctypes.POINTER(ctypes.c_int16)
```

Figure 11. The pointer to the first element is defined.

```
#Generate Z points
start = time.time()
surfaceDataPtr = GoSdk.GoUniformSurfaceMsg_RowAt(surfaceMsg, 0)
Z = np.ctypeslib.as_array(surfaceDataPtr, shape=(size,))
Z = Z.astype(np.double)
#remove -32768 and replace with nan
Z[Z==-32768] = np.nan
#scale to real world units (for Z only)
Z = (Z * ZResolution) + ZOffset
print("Z array generation time: ",time.time() - start)
```

Figure 12. A `numpy` array is generated from the pointer as length width*length (size). Points with a value of -32768 (invalid or no data) are changed to nan. Real world units are then calculated.

```
#write to file as np array (fast)
start = time.time()
unique_filename = str(uuid.uuid4())
np.save(unique_filename+"XYZ"+".npy",data_3DXYZ)
print("wrote to file "+unique_filename+ "XYZ" + ".npy")
print("Save npy file time: ",time.time() - start)

#write to CSV (slow)
start = time.time()
unique_filename = str(uuid.uuid4())
with open(unique_filename+"XYZ.csv",'w',newline='') as csvfile:
    writer = csv.writer(csvfile,delimiter=',')
    writer.writerow(["X","Y","Z"])
    writer.writerows(data_3DXYZ)
print("wrote to file "+unique_filename + "XYZ.csv")
print("Save CSV file time: ",time.time() - start)
```

Figure 13. Examples to write data to an `npy` file, and a CSV file. Normally commented out.

```
#Display the surface (it look square unless a perspective correction is done)
image = np.reshape(Z, (-1, width))
maxval = np.nanmax(image)
image = (image / maxval) * 255.0
image = image.astype(np.uint8)
image = cv2.resize(image, dsize=(512, 512), interpolation=cv2.INTER_CUBIC)
cv2.imshow("image", image)
cv2.waitKey(0)
```

Figure 14. Example to show the surface image. It must be downsampled due to Windows not being able to natively support 16 bit grayscale. Surface “pixels” are generally not square so some distortion is expected.

Uniform Intensity Image

Uniform Intensity images can be acquired in much the same way as Surface data. Instead of a 16-bit data type, it uses an 8-bit data type. Since the data is intensity, you don't need to scale it as with Surface data.

```
#resolutions and offsets (cast to mm)
XResolution = float((GoSdk.GoSurfaceIntensityMsg_XResolution(surfaceIntensityMsg)))/1000000.0
YResolution = float((GoSdk.GoSurfaceIntensityMsg_YResolution(surfaceIntensityMsg)))/1000000.0
XOffset = float((GoSdk.GoSurfaceIntensityMsg_XOffset(surfaceIntensityMsg)))/1000.0
YOffset = float((GoSdk.GoSurfaceIntensityMsg_YOffset(surfaceIntensityMsg)))/1000.0
width = GoSdk.GoSurfaceIntensityMsg_Width(surfaceIntensityMsg)
length = GoSdk.GoSurfaceIntensityMsg_Length(surfaceIntensityMsg)
size = width * length
```

Figure 15. Values from intensity image.

```
surfaceIntensityDataPtr = GoSdk.GoSurfaceIntensityMsg_RowAt(surfaceIntensityMsg, 0)
I = np.array((surfaceIntensityDataPtr[0:width*length]), dtype=np.uint8)
```

Figure 16. Intensity values pulled into a numpy array. Saving and displaying is the same process as with uniform surfaces.

Refer to Figures 13 and 14 for screenshots of saving and displaying intensity values.

Uniform Profiles

Uniform Profiles are 2D representations of resampled data. The axes of interest are X and Z where Z is a 1D array of distances, and X is represented by the index of each Z value.

```
#resolutions and offsets (cast to mm)
XResolution = float((GoSdk.GoResampledProfileMsg_XResolution(profileMsg)))/1000000.0
ZResolution = float((GoSdk.GoResampledProfileMsg_ZResolution(profileMsg)))/1000000.0
XOffset = float((GoSdk.GoResampledProfileMsg_XOffset(profileMsg)))/1000.0
ZOffset = float((GoSdk.GoResampledProfileMsg_ZOffset(profileMsg)))/1000.0
width = GoSdk.GoProfileMsg_Width(profileMsg)
size = width
```

Figure 17. Resolutions and offsets are pulled, similar to Uniform Surfaces.

```
#Generate Z points
start = time.time()
profileDataPtr = GoSdk.GoResampledProfileMsg_At(profileMsg, k)
Z = np.ctypeslib.as_array(profileDataPtr, shape=(size,))
Z = Z.astype(np.double)
Z[Z==-32768] = np.nan
Z = (Z * ZResolution) + ZOffset
print("Z array generation time: ",time.time() - start)
```

Figure 18. Z values cast to real world units in a numpy array.

```
#write to file as np array (fast)
unique_filename = str(uuid.uuid4())
np.save(unique_filename+"XZ"+".npy",data_3DXZ)

#write to CSV (slow)
unique_filename = str(uuid.uuid4())
with open(unique_filename+"XZ.csv",'w',newline='') as csvfile:
    writer = csv.writer(csvfile,delimiter=',')
    writer.writerow(["X","Z"])
    writer.writerows(data_3DXZ)
print("wrote to file "+unique_filename)
```

Figure 19. X and Z values saved to file.

Unsupported Messages

As of the writing of this document, the following common messages have yet to be implemented:

GoSurfacePointCloudMsg

GoMeshMsg

GoHealthMsg

GoVideoMsg

GoGenericMsg (blob and segmentation arrays)

For assistance, please contact LMI support at support@lmi3d.com