

Title: Gocator 4.x/5.x Multi-Sensor Setup and Control Using SDK
Revision: 1.2

NOTE: As of Gocator firmware version 4.6, multi-sensor systems can be set up and aligned directly in the web interface. This application guide, which describes setting up and controlling such systems using the SDK, is therefore no longer the only method for multi-sensor setup and control.

Table of Contents

1 Introduction	2
2 Setup	3
2.1 Synchronization	4
2.2 IP Configuration	5
2.3 Triggering	5
2.4 Exposure	6
2.5 Gocator Configuration	6
3 Multiplexing Rules for the Gocator	7
3.1 Simplified Exposure Rules	7
3.2 General Rules for Single Exposure	8
3.3 Dynamic Exposure Rules	9
3.4 Multiple Exposure Rules	9
3.5 Encoder Trigger Mode	9
4 Control Using the Gocator SDK	9
4.1 Manual Configuration	9
4.2 Automatic Configuration	10
4.3 Synchronously Start Sensors	10
4.3.1 Immediate Start Method	11
4.3.2 Scheduled Start Method	11
4.4 Match Data and Handle Dropout	11
4.5 Single vs. Multi-threaded Data Callback	12
4.5.1 Version 4.1+ Multi-threaded Data Callback Implementation	12
4.5.2 Version 4.0 Multi-threaded Data Callback Implementation	12
4.6 Process Data	13

1 Introduction

This guide explains how several Gocator sensors can be used together in a synchronized system in order to scan large objects at high resolution. The Gocator's built-in Buddy concept can handle most dual-sensor configurations, but when creating systems with three or more sensors the user has to work with the Gocator Software Development Kit (SDK) to build a custom solution. In high-speed applications with dual-sensor configuration, there may also be a need to use an SDK solution because Buddy mode is limited in speed due to CPU load.

The simplest form of multi-sensor system is when the sensors scan independent surfaces of a target, where there is no interference from overlapping measurement zones. In this case there is no need to multiplex the exposure between sensors and there may not be a need to synchronize the data collected either.

However, a common requirement with multi-sensor systems is to precisely control the timing of each sensor to avoid interference in areas where the measurement zones overlap, often referred to as "crosstalk" between sensors. The solution is to identify groups of sensors that are independent of each other (that is, without crosstalk issues) and logically group them into *banks*. Measurements in the system are then orchestrated in such a way that each bank takes turns exposing the object, that is, the banks are *multiplexed*.

The Gocator SDK has API level support to automatically configure the timing for a system with any number of sensors with any number of banks through the *GoMultiplexBank* class. It's also possible to program multiplex timing manually by following the rules outlined in this document.

Multi-sensor systems also often require a system calibration procedure in order for all sensors to report data in a common coordinate frame. A calibration target of known shape and dimension is positioned in the system allowing for a calibration procedure to calculate translation parameters for each sensor. During run-time the translation parameters are then applied to the data points to create a single 3D model of the scanned object.

This document focuses on the Gocator SDK and multiplexing rules, whereas the system calibration topic is too broad to describe in this document. The content is divided into three sections.

- **Setup** describes how to physically connect the sensors in a network and how to configure the sensors using the Gocator's web-based user interface. A connection to the sensors is made through a web browser on any operating system. No driver installation required.
- **Multiplexing Rules** explains the detailed procedure for determining the exact timing of each sensor in a network of Gocators to avoid crosstalk. A simplified case, as well as the theoretical general case, is covered. Both scenarios come with practical examples.
- **Gocator SDK** outlines how the user can implement custom multiplex timing in software using the Software Development Kit. This guide is bundled together with an open source example code project that the user can study, compile and run in parallel to reading this section.



2 Setup

There are many potential Gocator applications that require three or more sensors, and there are almost as many variations on the physical layout of such systems. Figure 1 shows a couple of real world examples.

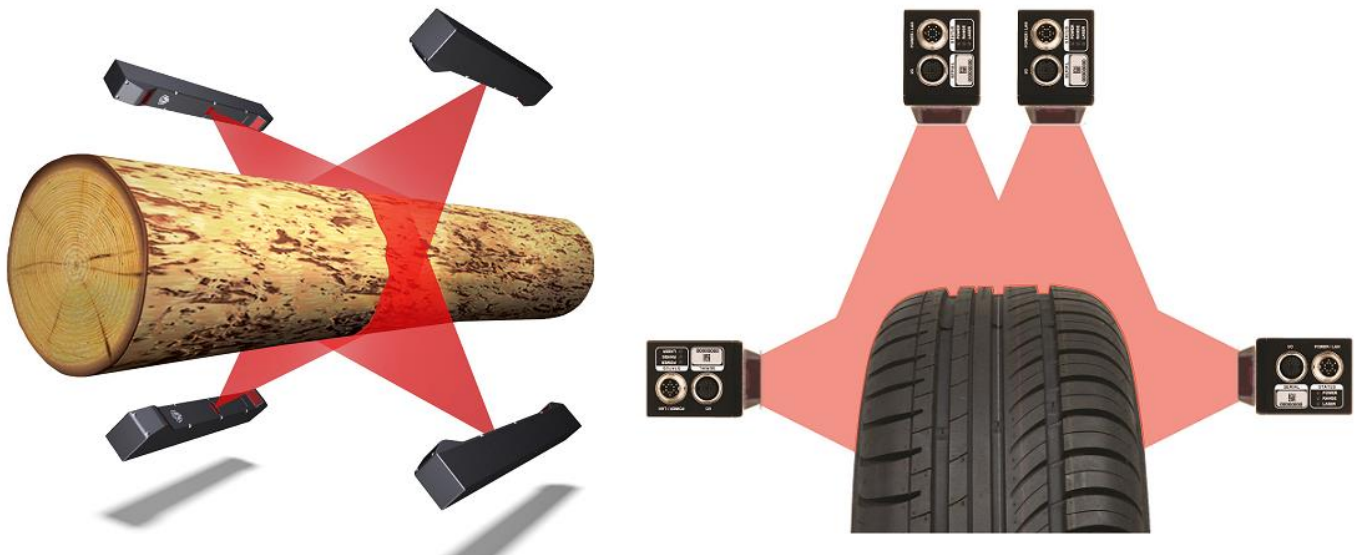


Figure 1: Multi-sensor applications. Log scanning (left) and tire scanning (right).

In the log-scanning example above with four sensors, neighbouring sensors are suffering from crosstalk. This means that the four sensors need to be logically divided into two banks, each consisting of two diagonally opposing sensors. As an example, assume that the sensors are operating at 100 Hz. This means that the time to acquire a single scan, referred to as the *frame period*, is 10 milliseconds. Multiplexing will then be achieved by splitting the 10 ms frame period into two halves, each 5 milliseconds, giving one independent time slot for each bank. Figure 2 shows the system configuration and timing diagram of this simple example. Opposing sensors A and C belong to Bank 0, whereas sensors B and D belong to Bank 1.

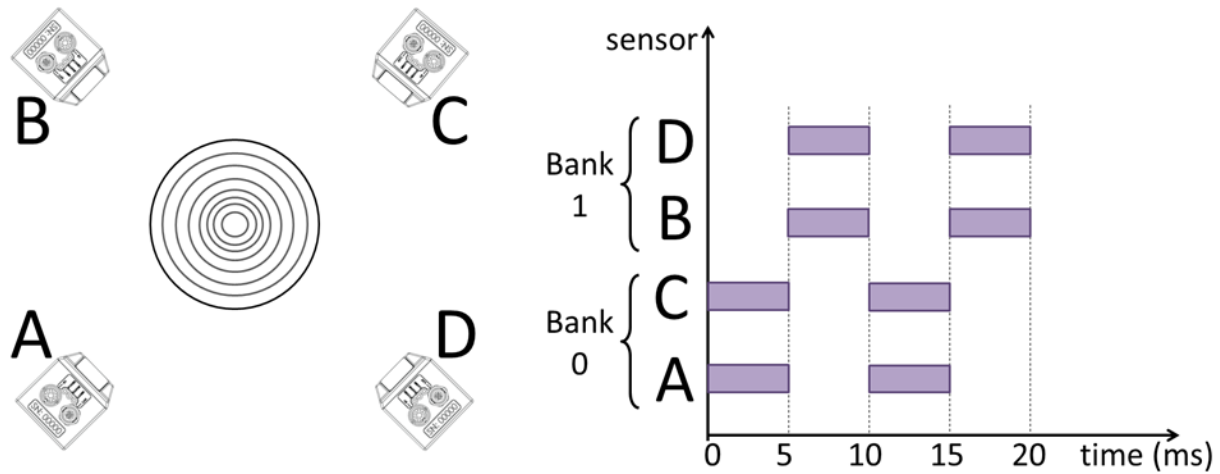


Figure 2: Example timing diagram for log-scanning application

2.1 Synchronization

Gocators are networked through a Master and an Ethernet switch. The Master accepts encoder and external input and provides power, laser safety, and synchronization to the Gocators. The synchronization information is encoded by the Master into a protocol that each Gocator decodes to recover time, encoder, and input I/O status. The Ethernet switch provides the data communication paths for connecting Gocators to the client computer.

For a multi-sensor system, Master 400, 800, 1200, or 2400 can be used. These Masters support the same functionality and are different only by the number of sensor connections supported.

A multi-system system is connected as follows:

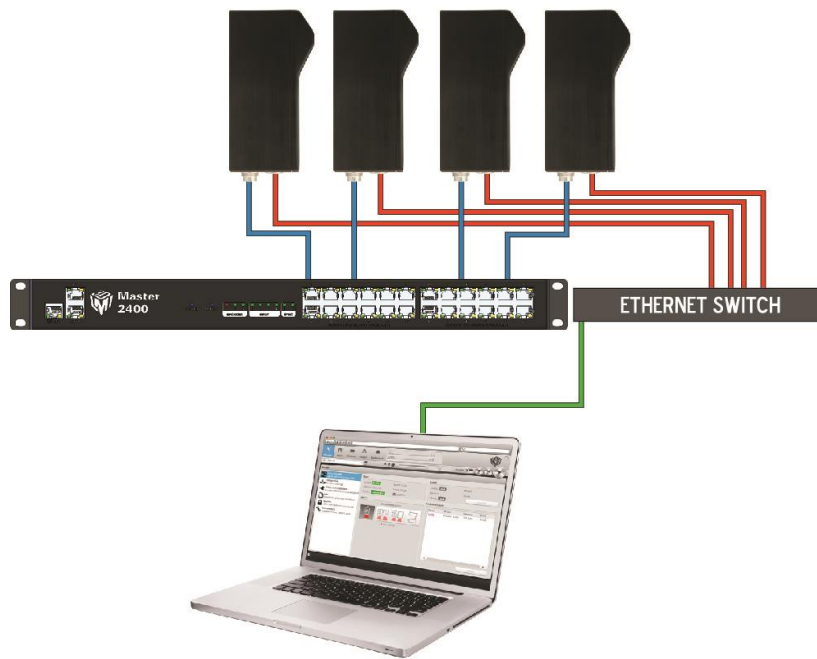


Figure 3: Gocator cable connection

Refer to section *Master 400/800* in the Gocator user manual on how to connect power, laser safety, and encoder signals to the Master 400. It's very important to ensure that the shield of the Gocator cord set cable is properly connected to earth ground to avoid electrostatic discharge.

2.2 IP Configuration

All Gocators are shipped with a default IP address of 192.168.1.10. Before connecting the Gocator to the Ethernet network, each Gocator requires a unique IP address.

Refer to the *Connecting to a New Sensor* section in the user manual on how to change the IP address of a Gocator.

2.3 Triggering

Triggering in a multi-sensor system is typically either time-based or encoder-based and is provided by the Master. With time-based scanning, the sensors are simply free-running at the specified speed. With encoder-based scanning, the sensors acquire data at the specified distance interval. In either case, the delivered profile data is always tagged with both a timestamp and an encoder stamp. These stamps can be used to resample data from many sensors that capture the data at slightly different times to a common spacing interval.

2.4 Exposure

Gocator supports three different exposure modes: Single, Dynamic, and Multiple. For multiplexing to work, all sensors must use the same exposure mode, and in the case of Multiple exposure, all sensors must use the same number of exposure steps.

2.5 Gocator Configuration

In a multi-sensor system, each Gocator is configured independently by logging in through a web browser using the Gocator's unique IP address.

To allow testing and evaluation of exposure multiplexing without having to write software with the SDK, the Gocator web interface can be used to quickly configure the timing. These settings are only shown in an advanced view by adding “?advanced=1” to the URL, for example: <http://192.168.1.10/?advanced=1/>.

The exposure multiplexing settings are on the **Manage** page under the **Sensor System** area, as shown below in Figure 4. The user can configure the delay from the beginning of the frame period before exposure starts, as well as the total period (which cannot be shorter than the minimum period as determined by general sensor configuration such as active area, sub-sampling, etc.).

It's important to understand the following points about exposure multiplexing:

- In a system of three or more sensors, the Gocators are *not* configured to operate in Buddy mode.
- When using time-based triggering, all sensors in a synchronized system *must* run at the same time period (speed).
- When using time-based triggering and when exposure multiplexing is enabled, the sensors will run at the speed defined by the **Time Period** setting, and *the Frame Rate setting in the Trigger panel on the Scan Page will be ignored*.
- When using encoder -based triggering, all sensors in a synchronized system must be configured with the exact same encoder resolution and the exact same trigger spacing.
- The time and encoder stamps of the data from the different sensors are taken *before* any exposure delays. In other words, all data that was acquired as the result of a particular trigger event has the identical time and encoder stamp, even if the exposure started at different times after the trigger. This makes it easier to match frames from different sensors, and because the user knows how the exposure delays are configured, it's possible to calculate the time/position of the start of exposure if needed.



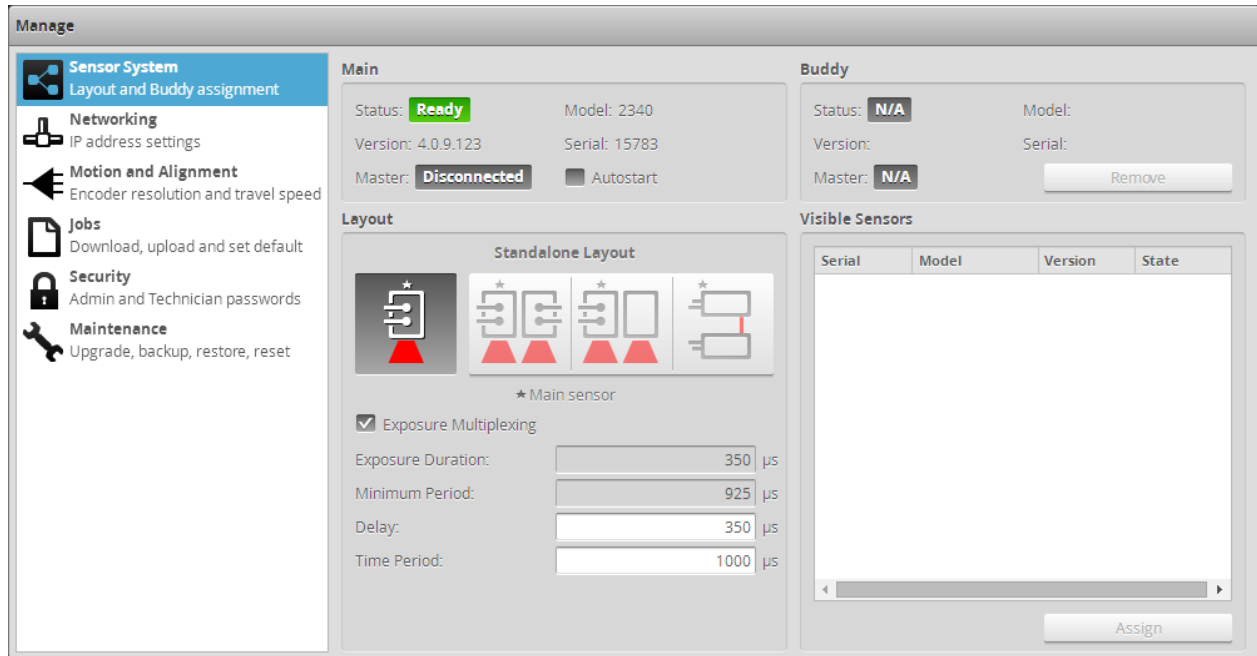


Figure 4: Configuring multiplexing in Gocator web interface.

3 Multiplexing Rules for the Gocator

As described in the introduction, the basic principle to avoid sensor crosstalk is to use a multiplexing strategy. Sensors that can operate independent of each other, without risk of crosstalk, are grouped into logical banks. The overall system speed determines the length of the time period for one data acquisition cycle, the so-called frame period. This frame period is then split up into shorter time slots, one for each bank of sensors to operate within. In practice, this means that multiplexing is achieved by aligning the exposure of each bank such that the exposure periods don't overlap, as explained by Figure 2 on page 4. The **Exposure Delay** parameter in the sensor configuration is the key mechanism to achieve various multiplexing behaviour

For the user interested in manually configuring the exact multiplex timing for every sensor in a larger system, use the rules described in this section. *Note that these rules are automatically handled by the GoMultiplexBank class.*

3.1 Simplified Exposure Rules

In most applications all sensors in the system are scanning the same type of target surface, for example, a log, the rubber of a tire, a road surface, etc. The characteristics of the surface typically determine the minimum exposure time needed for a reliable measurement and *all sensors in the system are running with the same exposure time.*

In most applications there is also a certain speed at which the system has to run in order to solve the problem at hand. This required speed then sets the available frame period.

In this simplified scenario the exposure rule is easy. The exposure time cannot be longer than: (frame period / number of banks).

The actual multiplexing is then achieved by delaying the start of the exposure for each bank.

Example

A system of three Gocator 2080 sensors is scanning a log. The sensors are mounted in a ring with a 120° separation. Each sensor's field of view (FOV) is overlapping with both of its neighbours, causing crosstalk, meaning that three banks are required. The application requires the sensors to run at 300 Hz, which means that the frame period is approximately 3333 µs. According to the simplified rule, the exposure time for each sensor cannot be longer than 3333 µs / 3 banks = 1111 µs. With a small margin added, a safe upper limit for the exposure time in this system is 1100 µs.

The multiplexing in this system is then implemented such that sensor 0 is exposing in the time slot from 0 µs to 1100 µs, sensor 1 is exposing in the time slot from 1111 µs to 2211 µs, and sensor 2 is exposing in the time slot from 2222 µs to 3322 µs.

The example source code project included with this document shows how to use the Gocator SDK to accomplish this exact scenario.

3.2 General Rules for Single Exposure

In this theoretical general case, every sensor in the system may run with a *different* exposure time. Although this scenario is less common in multi-sensor systems, it nevertheless provides a detailed understanding for how the Gocator operates. The procedure for establishing settings for multiplexing in this case is:

- Determine the desired exposure time for each sensor in the system.
- Determine how many multiplexing banks are needed.
- Within each bank, make note of the longest exposure time.
- Determine the exposure delay for each bank, such that the delay for a bank is the sum of the longest exposure in each preceding bank. For example, the exposure delay for Bank 2 should be the longest exposure in Bank 0, plus the longest exposure in Bank 1.
- Apply the exposure time and exposure delay to each sensor in the system. These settings will affect the maximum speed at which the Gocator can run. When the exposure settings are applied, the Gocator will automatically calculate this maximum speed and update a property called Maximum Frame Rate.
- Determine the overall frame rate the system can run at. The system will not be able to run faster than the slowest sensor, so query the Maximum Frame Rate property from each sensor in order to determine the slowest maximum frame rate. To *guarantee* that there is no crosstalk in the system, add a safety margin of 10 µs to the frame period. That is, invert the slowest maximum frame rate to get the period, and then add the 10 µs margin and then invert this result back to a frame rate. This is the fastest overall frame rate the system can run at while all the desired exposure times.

Example



In a system of 7 Gocator 2030 sensors, it is determined that 3 banks are needed to avoid sensor crosstalk. Application testing has determined that an exposure time between 500 μs and 1000 μs is required to get a good measurement off the target. The longest exposure time in Bank 0 is 750 μs , in Bank 1 it is 900 μs , and in Bank 2 it is 600 μs . This will give an exposure delay of 0 μs for all sensors in Bank 0, 750 μs for Bank 1, and 1650 μs for Bank 2.

All the exposure times and exposure delays are applied to the 7 sensors, and each sensor is then queried for the Maximum Frame Rate. In theory, the slowest sensors in the system should be the one with the 600 μs exposure time in Bank 2, as it has the longest exposure delay of 1650 μs , giving a total frame period of 2250 μs . This corresponds to approximately 440 Hz, but the result of the query is that the slowest sensor's maximum rate is reported as 300 Hz. The reason for this discrepancy is that the camera in the Gocator 2030 cannot run faster than 300 Hz at full measurement range. The Gocator can be configured to run much faster by manipulating the **Active Area** settings to narrow down the measurement range, but that's a detailed discussion outside the scope of this document.

3.3 Dynamic Exposure Rules

The rules for multiplexing a system of sensors that are running Dynamic exposure are identical to the rules for Single exposure, with one exception. The upper bound limit for dynamic exposure should be used in place of the single exposure in the rules outlined above in section 3.2.

3.4 Multiple Exposure Rules

When creating a multiplexed system of sensors that use the Multiple exposure mode, all sensors *must* use the same number of exposure steps. The total frame period of each individual sensor in this mode is *not* the sum of the exposures of all exposure steps. The length of the total frame period depends on a number of internal factors that are too detailed to discuss in this document. Instead the user should simply configure the exposure steps and times in the sensor and then query the Maximum Frame Rate property from the sensor. From this point the total frame period can be considered as the single exposure time, and the rules are again the same as for Single exposure outlined above in section 3.2.

3.5 Encoder Trigger Mode

The calculations outlined above are the same for encoder trigger mode. Exposure and exposure delays are always specified in time. The user only needs to ensure that the *encoder spacing is longer than the frame period*.

4 Control Using the Gocator SDK

The multi-sensor system can be configured and controlled using the Gocator SDK. Commands to start, stop, receive data, and so on are all available through an API. In fact, everything that can be done in the web based UI can also be done programmatically via the SDK.

An example project is included with this technical note. Users can reference this project as a template to start developing their application.

4.1 Manual Configuration

The user can handle banks and multiplex timing manually, applying the rules described in this document. The exposure delay and time period settings can either be manually entered through the web interface, as



described above in section 2.5, or through SDK calls on the *Layout* object. The *Layout* object is accessed through the *Setup* object, which in turn is accessed through the *Sensor* object.

Here is a pseudo-code example:

```
setup = GoSensor_Setup(sensor); //get setup handle from sensor object
layout = GoSetup_Layout(setup); //get layout handle from setup object
GoLayout_EnableMultiplexSingle(layout, kTRUE); //enable multiplexing (non-buddy)
GoLayout_SetMultiplexSingleDelay(layout, 350); //delay this sensor's exposure 350us
GoLayout_SetMultiplexSinglePeriod(layout, 1000); //set time period to 1000us
```

It's worth noting again that if multiplexing is enabled, it's this multiplex period that determines the speed the sensor runs at, and the trigger frame rate (which otherwise controls speed) is ignored.

To see a code example for manual configuration and how to receive data when it's not required to match data between the sensors (that is, when each sensor scans an independent surface), refer to the *GoMultiSensor_BasicSetup* example in the included source code project.

4.2 Automatic Configuration

The Gocator SDK has API level support to automatically configure the timing for a system with any number of sensors with any number of banks through the *GoMultiplexBank* class. The general workflow is described below:

1. Connect to the sensors to multiplex with (*GoSensor_Connect* for individual sensor or *GoSystem_Connect* for all sensors in system).
2. Set up each sensor's exposure time if needed.
3. Call *GoSystem_AddMultiplexBank* to add X number of multiplexing banks.
4. For each *GoMultiplexBank* handle returned by the above call, call *GoMultiplexBank_AddSensor* to add selected connected sensors to each bank.
5. Once all multiplex banks are defined, call *GoSystem_UpdateAllMultiplexParameters* with a period argument of 0.0 to have all sensors' multiplexing-related configuration elements automatically updated to run at the fastest possible speed.
 - a. The call to *GoSystem_UpdateAllMultiplexParameters* implicitly enables multiplexing by calling *GoLayout_EnableMultiplexSingle*.
 - b. The call to *GoSystem_UpdateAllMultiplexParameters* automatically sets the exposure delays and time period for all sensors using the *GoLayout_SetMultiplexSingleDelay* and *GoLayout_SetMultiplexSinglePeriod* calls.
6. To start all sensors in the system use *GoSystem_Start*.

To see a code example using this work-flow, refer to the *GoMultiSensor_AutomaticSetup* example in the included source code project.

4.3 Synchronously Start Sensors

As long as a multi-sensor system is connected through a Master, and all sensors use the same time period, the sensor acquisition cycle is always synchronized. This means that no matter when each sensor is started the acquisition between sensors is aligned to microsecond precision, and all data that was acquired as the result of a particular trigger event has the identical time and encoder stamp.



A sensor can be individually started through the `GoSensor_Start` call, but there are also two methods to start the full system.

4.3.1 Immediate Start Method

The `GoSystem_Start` call is an Immediate Start method where the system starts each sensor as soon as possible. With this method the user does not need to worry about how long it may take from the time the command is issued from the client application until the sensor is ready to start.

However, this means that the `GoSystem_Start` call does not guarantee that all sensors start in the same acquisition cycle. If the user needs to exactly match data from all sensors that were acquired at the same time/position, it is not possible to blindly match based on the order of incoming data. The user must create code that discards initial data until all sensors are up and running and delivering data.

To see a code example to handle this initial state, refer to the `GoMultiSensor_AutomaticSetup` example in the included source code project.

4.3.2 Scheduled Start Method

With the Scheduled Start method the user must specify a future start time/position at which point all sensors are guaranteed to start at the exact same time. In this case the user must query a sensor to find the current time/position and then add enough of a margin to the future time/position to ensure all sensors have received the command and are ready to start.

The benefit of this method is that *if the user can guarantee that the sensor configuration and trigger timing does not cause any data drop*, then the frames from the different sensors can be blindly matched by frame count. To handle data drop, see section 4.4 below. Another benefit of the Scheduled Start method is that in case of very slow sensor speed (for example, 1 Hz and slower) the system can start faster than with the Immediate Start method.

Note: With Gocator firmware release 4.0 and 4.0 Service Release 1, there is no Scheduled Start command as in 3.x firmware. The Scheduled Start command will be re-introduced in Gocator firmware 4.1.

4.4 Match Data and Handle Dropout

In some multi-sensor applications, it is a requirement to exactly match data from all sensors that were acquired at the same time/position. This includes handling the potential initial frame mismatch with the Immediate Start method, but also matching multiplexed data, where data from the same frame can come from a different time/position from different sensors, and also ensuring matching does not get out of sync if data is unexpectedly dropped from a sensor.

A typical implementation to fully match data from multiple sensors will use a queue for each sensor buffering the incoming data. Matching logic will analyse the time/encoder stamps on the output of the queues.

In terms of send order in a system of Gocators, there is only one guarantee: data from one specific sensor is sent in the order of acquisition. However, between sensors, data can be delivered in any order. For example, in a system of 3 sensors (A, B, and C), the client might receive a sequence like this: B1, B2, B3, C1, A1, A2, A3, C2, C3, B4, and so on.



To see a code example showing queues and matching logic, please refer to the *GoMultiSensor_AutomaticSetup* example in the included source code project.

4.5 Single vs. Multi-threaded Data Callback

The Gocator SDK includes support for receiving data asynchronously with a callback in a separate thread. In the normal use case with multiple sensors, the user would add all sensors in the system to the *System* object and then call *GoSystem_SetDataHandler* to create a single callback thread receiving data from all sensors. Inside the callback, the user would extract the sensor ID with the function *GoDataSet_SenderId* from the incoming data message to identify from which sensor the data came. This is the intended design and use of the *System* object. With this design, if the user needs separate processing threads for each sensor, the user will have to create those threads in his or her own code and distribute the data into them from the callback thread.

4.5.1 Version 4.1+ Multi-threaded Data Callback Implementation

In Gocator firmware version 4.1 and above, new function *GoSensor_SetDataHandler* is introduced to allow users to pass different callback function pointer to be used by each individual sensor when data is received. This automatically creates a new thread per sensor while all sensors are added in the system to the *System* object. The method described in Section 4.5.2 of this guide is obsolete with Gocator SDK and firmware version 4.1 and above.

Following is pseudo-code example:

```
GoSystem_Construct(&system, kNULL); //construct system object
GoSystem_FindSensorByIpAddress(system, &ipAddress, &sensor1); //create sensor 1
GoSystem_FindSensorByIpAddress(system, &ipAddress, &sensor2); //create sensor 2
GoSensor_SetDataHandler(sensor1, onData1, context1); //set callback for sensor 1
GoSensor_SetDataHandler(sensor2, onData2, context2); //set callback for sensor 2
GoSystem_Connect(system); //open command channel to all sensors in the system
GoSystem_EnableData(system); //open data channel for all sensors in the system
GoSystem_Start(system); //send start command to system
```

To see a code example showing how to multithread the callbacks with *GoSensor_SetDataHandler*, refer to the *GoMultiSensor_Multi-sensorthread* example in the included source code project

4.5.2 Version 4.0 Multi-threaded Data Callback Implementation

In version 4.0, it's also possible to create a *System* object for each sensor, which then can be associated with different callback threads for each sensor. The *GoSystem* calls only act on connected sensors, but because *GoSystem_Connect* connects to all sensors on the network, it *cannot* be used in this scenario. Instead, the *GoSensor_Connect* call should be used to connect to the individual sensor, and then data from only that sensor will be passed to a callback in a sensor-specific thread.

Here is a pseudo-code example:

```
GoSystem_Construct(&system1, kNULL); //construct system object 1
```



```

GoSystem_Construct(&system2, kNULL); //construct system object 2
GoSystem_FindSensorByIpAddress(system1, &ipAddress, &sensor1); //create sensor 1
GoSystem_FindSensorByIpAddress(system2, &ipAddress, &sensor2); //create sensor 2
GoSystem_SetDataHandler(system1, onData1, context1); //set callback for sensor 1
GoSystem_SetDataHandler(system2, onData2, context2); //set callback for sensor 2
GoSensor_Connect(sensor1); //open command channel for sensor 1
GoSensor_Connect(sensor2); //open command channel for sensor 2
GoSensor_EnableData(sensor1); //open data channel for sensor 1
GoSensor_EnableData(sensor2); //open data channel for sensor 2
GoSensor_Start(sensor1); //send start command to sensor 1
GoSensor_Start(sensor2); //send start command to sensor 2

```

To see a code example showing how to multithread the callbacks, refer to the *GoMultiSensor_Multi-datathread* example in the included source code project

4.6 Process Data

Data processing itself is not discussed in detail in this document. It can vary dramatically depending on the nature of each particular application, but the basic steps are often:

- Data buffering and memory management
- Transformation of profile data into world coordinates
- Resampling to an even coordinate grid
- Application-specific algorithms to produce final results

Note that when configuring a multi-sensor system to multiplex the data capture, the data is not acquired from the exact same location in the direction of travel (Y). The standard solution to this problem is to employ a resampling strategy to an even interval in Y that is the same across the whole system.

