LMI TECHNOLOGIES

Gocator Firmware 5.3.33.39 – Release Notes

Firmware Version 5.3.33.39 for G200 sensors
Document Revision A

Compatibility

- Devices Supported
  - Gocator 210, 230, 240 and 250 Multi-Point Sensors
  - Gocator 205 Vision Sensors
- Web Interface
  - The web interface requires Chrome, Firefox or Microsoft Edge version 79 or later.
  - Old Edge versions are only supported in a limited fashion (see known issues for details).
- Backwards compatibility:
  - 5.3.33.39 GoWebScanSDK is compatible with applications using older 5.x SDK and firmware
  - 5.3.33.39 sensor firmware is compatible with older 5.x SDK and applications
  - 5.3.33.39 example application is **not** compatible with older 5.x SDKs
  - For users who have customized the SDK, please compare all files in the SDK with your customized SDK since the changes in this release are extensive.

Improvements

Support for G200 C Revision	Updates for the C revision G200 series hardware support a new maximum frame rate of 4 kHz, OR an extended measurement range of 10”. C revision sensors that have a firmware prior to 5.3.33.26 cannot run at 4 kHz or over a 10” range, but will still operate at the previous settings. The default settings for frame rate and measurement range remain 3 kHz and 8”, to assist installation in existing G200 systems.
Increasing sensor transmit rate	To sustain an increased data output rate (either through running at 4 kHz or at 3kHz with x subsampling disabled), support has been added to increase each sensor’s transmit limit (expressed as a percentage of a sensor's link capacity). Functions have been added to read, write, and calculate the transmit limit. The example program has been updated to demonstrate usage of these functions.
Increase to 10 inch Measurement Range	New G200C sensors can operate over the new measurement range of 10” at 3 kHz. Calibration has been done over an extra inch at the near and far end of the standard 8” range, allowing installation into existing G200 systems without requiring changes to sensor standoff. Increasing the measurement range to the new maximum of 10” is done using existing active area functions in GoWebScanSDK.

+1 604 636 1011
+1 604 516 8368
info@lmi3D.com
www.lmi3D.com
HEADQUARTERS
9200 Glenlyon Parkway
Burnaby, BC V5J 5J8
Canada

System Timing Verification	An optional function has been added to GoWebScanSDK that verifies system timing. This function can be added to applications to verify that the current system timing does not result in interference between sensors.
Update to Vision Delay Shift Calculation	Some of the multiplexing timing calculation logic has been moved from the example program into the SDK as a utility function.
Offline Replay	Added support for offline replay functionality. There is also a new dedicated class for recording files, GoWebScanRecording. This has replaced the previously used recording class instances (kArrayLists) in both GoWebScanProcess (where recording can be enabled for detection or web mode), and in the example program (where a recording is loaded from disk to replay).
Output of Calibration Bar Scans	The example program has added functionality to produce a detection or web output of the calibration bar or produce a raw recording. This works by the SDK reusing the raw calibration bar sensor data as board data.
Improvements to Example Program	The example program was rewritten to improve usability and clarity. The program consists of a main menu with submenus for the different functionalities. One can return to the main menu after performing a calibration or detection instead of restarting the program. The example program includes functionality to update the firmware of all connected sensors simultaneously. The storage folder can be set by an optional argument when running GoWebScanSdkExample.exe. Output board data is now saved into a "data" subfolder within the storage folder.
Web Mode	Added support for Web Mode. This mode compiles the tiles generated in detection mode into a complete grid for each board, leaving board detection logic to the user.
Multi Station Support	Added capability to align calibrations from multiple stations to a common reference.
Calibration Import/Export from String	Added ability to save and load calibration from a string containing the text in the Calibration.xml file.
System Mode Switching	Updated *GoWebScanConfig* with new functions that update system mode and calibration without requiring a restart.
Callback Functions for Output Data	Added callback functions for applications to process completed calibration and board data.
Callback functions for when camera lanes are enabled or disabled	A callback function can be set for when a camera lane is disabled after not receiving data for an extended period of time or distance, and for when it is re-enabled again.
Moved VIEW_SPOT_COUNT macros to the SDK	Added new definitions for VIEW_SPOT_COUNT macros in the SDK, such that applications can be simplified. This is an optional change.

Bug Fixes

Duplicate IP Address Issue Resolved	Previously sensors with duplicate IP addresses on the same network were inaccessible and/or exhibited unpredictable behaviour. This has been fixed (sensor firmware must first be updated to 5.3.33.39).
Calibration files with errors	Calibration files are no longer used by the application when calibration fails due to an error.
Improved data recording speeds	Data recording has been optimized for speed.
Corrected XCentre units	The XCentre values in the sample Settings.xml file now correctly specify in mils, rather than inches.
Recording crashes	Fixed crash which could occur if recording is replayed with incorrect sensor info.
Use Vision Serial for Vision Warnings	Vision warnings during calibration now display the vision serial number rather than the profile serial number.
Address Duplicate Colour Gain Warning	Warnings for the x coverage are now only displayed once per camera.

Known Issues

Translations Incomplete	Not all English text is translated in every language.

Detailed Release Features

These notes detail all changes, improvements and new features for GoWebScanSDK that have been incorporated since the release of GoWebScanSDK 5.3.32.4.

This release adds support for G200C revision sensors which can operate at faster speeds and larger measurement ranges. It also includes web mode support, system timing verification, calibration import/export via string, callback functions for when camera lanes are enabled and disabled, callback functions for when calibration or board data is ready to be processed, calibration alignment among multiple systems, asynchronous firmware update functionality, and several changes to support reprocessing of calibration data in detection and web mode.

Also, the example program has been rewritten to demonstrate the new features in a more organized fashion.

Changes to SDK

- Support has been added to reprocess calibration data as a regular detection or web output. Once calibration is completed, the data making up the calibration may be retrieved with the *GoWebScanCalCollector_GetCalRecording* function. This data is stored in a *GoWebScanRecording* instance, which is the new format of data recordings that can be replayed (previously a kArrayList was used). One may save a calibration recording to disk then load it to be reprocessed in Calibration, Detection, or Web Mode.
- Support for Web Mode has been added. This mode outputs the tiles of the scan and leaves the board detection logic to the user.
- Support for changing modes has been simplified. One must call *GoWebScanConfig_UpdateMode* on the config with the desired mode. Similarly, the calibration can be changed with *GoWebScanConfig_UpdateCalibration*. After the config has been updated, *GoWebScanProcess_Refresh* must be called in the case of Detection or Web Mode. This is demonstrated in the example program.
- Calibration can be performed multiple times. One must call *GoWebScanCalCollector_Reset* and *GoWebScanCalProcessor_Reset* prior to each new calibration.
- In Detection or Web Mode, *GoWebScanProcess_SetDataHandler* can be used to set a callback function on board data when it is ready. Similarly, in Calibration Mode, *GoWebScanCalCollector_SetOnCompleteHandler* can be used to set a callback function for when the calibration collector is ready. Usage of both of these functions is demonstrated in the example program.

- Calibration originally was only loaded from or saved to an .xml file. It can now be imported from or exported to a string in memory. This is done with the *GoWebScanCal_ImportFromText* or *GoWebScanCal_ImportFromString* and *GoWebScanCal_ExportToString* functions.
- *GoWebScanUtils_AlignCalibrations* has been added as a helper function to align the calibrations of multiple stations. The example program includes a demonstration of how it is used.
- *GoWebScanProcess_SetLaneDisabledHandler* sets a callback function for when a camera lane has been disabled after not receiving data for an extended period of time. Similarly, *GoWebScanProcess_SetLaneEnabledHandler* sets a callback function for when such a lane is re-enabled after receiving data again. The corresponding sensor and camera can be determined from the lane id using the *GoWebScanConfig_NodeColumnToIndex* function, and vice versa with the *GoWebScanConfig_NodeIndexToColumn* function. This is demonstrated in the example.
- *GoWebScanCalCollector_Flush* has been added to flush sensor data through the collector pipeline. Also, checking of frame drops can be disabled with *GoWebScanCalCollector_CheckFrameDrops*, and whether or not they are checked can be seen with *GoWebScanCalCollector_FrameDropsChecked*. These functions are only needed for replaying data in Calibration Mode.
- Sensor info can now be saved to and loaded from a folder with dedicated functions, *GoWebScanUtils_StoreSensorInfo* and *GoWebScanUtils_LoadSensorInfo*. A sensor info folder path can also be verified with *GoWebScanUtils_SensorInfoFilesExist*.
- *GoWebScanUtils_AsyncSensorFunction* can be used to asynchronously call a function on multiple GoSensor objects at once. Its usage is demonstrated in the example program which now has functionality to update the firmware of all connected sensors.
- Calibration vision gains were previously compressed upon being saved to disk, meaning a newly created calibration in memory would have been slightly different from one saved to then loaded from disk. They are now compressed on creation (when being set via the update functions: *GoWebScanCalNode_UpdateVisionRGains*, *GoWebScanCalNode_UpdateVisionGGains*, and *GoWebScanCalNode_UpdateVisionBGains*), so this discrepancy no longer exists. As part of this change, *GoWebScanCalNode* now has explicit getters and setters to data arrays (profile Y and Z offsets, vision Y offsets, and vision gains). These are internal functions to GoWebScanSDK and do not affect the example program.
- Instances of *GoWebScanSettings* can be cloned via kObject_Clone.
- Instances of *GoWebScanCal* can be compared with kObject_Equals.
- *GO_WEB_SCAN_VIEW_SPOT_COUNT_G250* and *GO_WEB_SCAN_VIEW_SPOT_COUNT_G210* have been defined in *GoWebScanSdkDef.h*, and may be used in place of *DEFAULT_VIEW_SPOT_COUNT_G250* and *DEFAULT_VIEW_SPOT_COUNT_G210* which were originally in the *GoWebScanSdkExample* code.

Changes to Example Program

- The example program has been rewritten. It now has a main menu, which has options to choose a custom storage folder, toggle replay, run one of the three modes (calibration, detection, or web mode), update the firmware, or quit. After choosing an option, one can generally return to the main menu and choose another option, without having to restart the program.
- The main menu displays all connected sensors and the versions of their firmware.
- A custom storage folder can be chosen. This can be done either from the main menu, or a storage folder may be specified as an argument when running GoWebScanSdkExample.exe.
- In the calibration submenu, once a calibration has completed successfully, there are options to output the calibration bar data as a detection or web mode output, either calibrated or uncalibrated. The calibration data may also be saved to disk and replayed as a recording.
- The calibration menu also has an option to align the calibrations of multiple systems.
- Output data from Detection and Web mode is now saved to a "data" subfolder within the storage folder.
- There is an option to upgrade the firmware of all connected sensors.
- The example program indicates when it has exited without any errors.

Bug Fixes

- Sensors with duplicate IP addresses on the same network behaved unpredictably. This has been fixed in the 5.3.33.39 firmware. Sensors with duplicate IP addresses can now be connected to the same network and users can modify the IP addresses via *GoSensor_SetAddress* in GoSDK.
- During calibration, vision warnings included the serial numbers of the respective profile sensor. This has been fixed to display the serial number of the vision sensor.
- During calibration, warnings with regards to x coverage could be displayed multiple times for the same sensor and camera. This has been fixed so it only displays the warning once per unique camera.
- Calibration files generated with an error should not be used. *GoWebScanSdkExample* has been modified to no longer save the Calibration.xml file in the event that calibration fails due to an error.

- Data recording functionality has been optimized to improve recording speeds.
- The example code for replaying a recording has been made faster. The previous example would wait 10 milliseconds for each *GoDataSet* processed in the replay function, in order to give time for an output to be produced. This has been removed, which makes replaying faster, but the program now requires an expected board output count to be provided.
- A memory leak has been addressed which could have slowed down the system.
- An issue was fixed in which data was sometimes unnecessarily dropped.
- The example XCentre values in the sample Settings.xml were specified in inches (values of 12 and 36) but have been corrected to mils (12000 and 36000).
- Fixed crash which could occur if recording is replayed with incorrect sensor info.

New Features

Many of the features in the following sections are demonstrated in the example program that is included in the GoWebScanSDK package. The user can reference this program for assistance when incorporating these into their own applications.

G200C and Backwards Compatibility

G200C revision sensors support a new maximum frame rate of 4kHz, OR an extended measurement range of 10”. To utilize the new functionality, firmware **5.3.33.26** or later is needed.

Older revision G200 sensors are compatible with the newer firmware, and can be run in a system alongside C revision sensors.

The C revision sensors can also be loaded with an earlier firmware version, and will utilize the existing default settings of 3 kHz scan rate and 8” measurement range. This is to facilitate installation into existing G200 systems.

Increasing the Transmit Rate

Support has been added to increase the frame rate to 4kHz. The increased frame rate results in an increased data output rate, such that the sensor transmit limits need to be increased as well.

Each sensor has its own transmit limit, a parameter which caps the sensor's output data rate in order to prevent burst transmissions. Transmit limit is expressed as a percentage of the individual sensor's link capacity, which is 1 gigabit per second.

Functionality has been added to read and write the transmit limit for an individual sensor, and to calculate the desired transmit limit. Using these functions is required in scenarios where the data rate is higher than typical usage, such as if the system is run at 4kHz, or if x subsampling is disabled at 3kHz.

GoSetup_SetTransmitLimit and *GoSetup_TransmitLimit* are functions to get and set the transmit limit.

GoWebScanUtils_CalculateProfileTransmitLimit and *GoWebScanUtils_CalculateVisionTransmitLimit* are functions to calculate the desired transmit limit of profile and vision sensors. These functions must be called after other settings have been applied to the sensor, since calculation depends on the setup of the sensor.

Profile sensor example:

```C/C++
if (!kSuccess(GoSystem_FindSensorById(context->system, profileId,
    &profileSensor)))

{

    printf("Error: SN %d not found\n", profileId);

    return kERROR;

}

setup = GoSensor_Setup(profileSensor);


// Apply any other sensor settings here.


kCheck(GoWebScanUtils_CalculateProfileTransmitLimit(context->settings,
    profileSensor, &item.profileTransmitLimit));

kCheck(GoSetup_SetTransmitLimit(setup, item.profileTransmitLimit));
```


Vision sensor example:

```
C/C++
if (!kSuccess(GoSystem_FindSensorById(context->system, visionId,
    &visionSensor)))

{

    printf("Error: G205 SN %d not found\n", visionId);

    return kERROR;

}

setup = GoSensor_Setup(visionSensor);


// Apply any other sensor settings here.


kCheck(GoWebScanUtils_CalculateVisionTransmitLimit(context->settings,
    visionSensor, &item.visionTransmitLimit));

kCheck(GoSetup_SetTransmitLimit(setup, item.visionTransmitLimit));
```

The *GoWebScanSdkExample* code has been updated to include this functionality.

Using 10 Inch Measurement Range

Support for hardware revision C sensors has been added, which are calibrated over an extended measurement range. By default, this range is 8 inches. To increase the measurement range to the full 10 inches, use the active area functions *GoSetup_SetActiveAreaZ* and *GoSetup_SetActiveAreaHeight*. Note that the values are in mm.

Example:

```
C/C++
if (!kSuccess(GoSystem_FindSensorById(context->system, profileId,
    &profileSensor)))

{

    printf("Error: SN %d not found\n", profileId);

    return kERROR;

}


setup = GoSensor_Setup(profileSensor);


kCheck(GoSetup_SetActiveAreaZ(setup, GO_ROLE_MAIN, -127));

kCheck(GoSetup_SetActiveAreaHeight(setup, GO_ROLE_MAIN, 254));
```


System Timing Verification

The new function *GoWebScanUtils_VerifySystemTiming* verifies that the system timing is one that does not result in interference between sensors. Usage of this is optional.

Example:

```
C/C++
// Read system settings

if (!kSuccess(GoWebScanSettings_Load(&contextPointer.settings,
    "..\\..\\storage\\Settings.xml", kNULL)))

{

    printf("Error: Unable to load GoWebScan settings\n");

    return kERROR;

}


// Verify the system settings

if (!kSuccess(status =
    GoWebScanUtils_VerifySystemTiming(contextPointer.settings, errorText,
    kCountOf(errorText))))

{

    printf("Error: %s\n", errorText);

    return status;

}
```

The *GoWebScanSdkExample* code has been updated to include this functionality.

Calculate Vision Delay Shift Update

Some of the multiplexing timing calculation logic has been moved from the example program into the SDK as a utility function. This includes function *GoWebScanUtils_CalculateVisionDelayShift*, which calculates the vision delay shift, and can optionally replace *calculateVisionDelayShift* which was originally in the *GoWebScanSdkExample* code.

Example:

```
C/C++
k32u calculateVisionStartDelay(DataContext* context, GoWebScanSettingsSensor
    sensorSettings)

{

     k32u periodMultiplier, delayShift, delay, visionExposureMax;

     // Maximally space top and bottom vision exposures to prevent interference
    with each other

     if (GoWebScanSettingsSensor_Plane(sensorSettings) ==
    GO_WEB_SCAN_SYSTEM_PLANE_TOP)

         periodMultiplier = context->visionFrameRateDivisor / 4;

     else

         periodMultiplier = 3 * context->visionFrameRateDivisor / 4;


     // Vision sensors expose in the empty region of the profile period to avoid
    interfering with profile sensors.


     //Old calculation

     //delayShift = context->separateTracheidExposure ?
    calculateVisionDelayShift(context) : context->profilePeriod / 2;


     //New calculation

     delayShift = GoWebScanUtils_CalculateVisionDelayShift(context->settings);

...
```


Lane Enabled/Disabled Callback Functions

A camera lane (either profile, vision, or tracheid) is disabled when it does not receive any data for an extended period of time or distance, and is re-enabled when it receives data again. *GoWebScanProcess_SetLaneDisabledHandler* and *GoWebScanProcess_SetLaneEnabledHandler* have been added to set callback functions for when such events occur. The example program uses these functions to print a message indicating which lane has had its status changed:

```
C/C++

...

        // Optional: Set handler for when camera lane is disabled due to
    receiving no data and exceeding time and distance coherency limits

        kTest(GoWebScanProcess_SetLaneDisabledHandler(context->process,
    laneDisabled, context));



        // Optional: Set handler for when camera lane is re-enabled after
    receiving data again

        kTest(GoWebScanProcess_SetLaneEnabledHandler(context->process,
    laneEnabled, context));

...
```

laneEnabled is defined as follows (with laneDisabled being defined in a similar manner):

```
C/C++
kStatus kCall laneEnabled(void* ctx, const k32s* id, kSize length)
{
    DataContext* context = (DataContext*)ctx;
    GoWebScanNodeTileId nodeTileId;
    GoWebScanSystemPlane plane;
    kSize column;
    kSize sensorIndex;
```


```
    kSize bankId;
    GoWebScanSettingsSensor sensor;
    k32u profileId;
    k32u visionId;

    kCheckArgs(length >= 4);

    nodeTileId = (GoWebScanNodeTileId)id[1];
    plane = (GoWebScanSystemPlane)id[2];
    column = (kSize)id[3];

    kCheck(GoWebScanConfig_NodeColumnToIndex(context->config, context->settings,
   plane, column, &sensorIndex, &bankId));

    sensor = GoWebScanSettings_SensorAt(context->settings, sensorIndex);
    profileId = GoWebScanSettingsSensor_ProfileSerialNumber(sensor);
    visionId = GoWebScanSettingsSensor_VisionSerialNumber(sensor);

    if (nodeTileId == GO_WEB_SCAN_NODE_TILE_ID_PRESENCE)
        printf("Presence lane enabled for sensor %d, bank %d.\n", profileId,
   (k32u)bankId);
    else if (nodeTileId == GO_WEB_SCAN_NODE_TILE_ID_PROFILE)
        printf("Profile lane enabled for sensor %d, bank %d.\n", profileId,
   (k32u)bankId);
    else if (nodeTileId == GO_WEB_SCAN_NODE_TILE_ID_TERRAIN)
        printf("Terrain lane enabled for sensor %d, bank %d.\n", profileId,
   (k32u)bankId);
    else if (nodeTileId == GO_WEB_SCAN_NODE_TILE_ID_VISION)
        printf("Vision lane enabled for sensor %d, bank %d.\n", visionId,
   (k32u)bankId);
    else if (nodeTileId == GO_WEB_SCAN_NODE_TILE_ID_TRACHEID)
        printf("Tracheid lane enabled for sensor %d, bank %d.\n", profileId,
   (k32u)bankId);

    return kOK;
}
```

Note the usage of the new function *GoWebScanConfig_NodeColumnToIndex* which can be used to determine the sensor and camera from the plane and column which are included in the lane id. A complementary function *GoWebScanConfig_NodeIndexToColumn* also exists, which can determine the column based on the plane, sensor index, and camera index.

Web Mode

The example program has been updated to demonstrate web mode usage (*GO_WEB_SCAN_MODE_WEB* in the SDK, an alternative to *GO_WEB_SCAN_MODE_CALIBRATION* and *GO_WEB_SCAN_MODE_DETECTION*).

Unlike detection mode which produces a whole board, web mode produces data in the form of continuous tiles, leaving the board detection logic up to the user. Note that this results in the immediate example program saving several small images to disk, rather than a few large ones. Web mode data is still affected by calibration and the chosen resolutions.

Multi Station Support

The example program includes a demonstration of how to align calibrations from multiple stations to a common X and Y reference. In a setup using multiple stations, calibration may be performed by running calibration mode in the example program on each of the stations, performing a scan of the calibration bar per station, and then aligning the resulting calibrations.

Alignment consists of determining the average system reference of all calibrations (in Calibration.xml > XReference and YReference), then setting the adjustment (in Settings.xml > Calibration > XAdjustment and YAdjustment) to the average system reference minus that station's system reference. This can be done with the *GoWebScanUtils_AlignCalibrations* function.

Note that the example program requires access to the calibration and settings files of the other stations to perform alignment. Once the alignment is complete, the X and Y origins of the board messages from each station will be with respect to a common X and Y reference position and can be used to place the data into a common buffer.

Calibration Import/Export from String

Support has been added to calibration files for saving to and loading from strings. The contents of the string consist of the text in the Calibration.xml file.

Outputting Calibration Bar Image after Calibration

Calibration in the example program has been modified so that once it completes, there will be options to produce calibrated or uncalibrated detection or web mode outputs of the calibration bar (with the same data used in creating the calibration).

Once calibration is complete, the detection or web output may be produced with the following steps:

1. Call *GoWebScanProcess_Clear* to clear the pipeline of any lingering messages.
2. Change modes in both the example and config via *GoWebScanConfig_UpdateMode*. In the example it is changed to *GO_WEB_SCAN_MODE_DETECTION*.
3. Change the config calibration. In the example it is changed to the newly created calibration, but it may also be changed to kNULL for an uncalibrated detection.
4. Call *GoWebScanProcess_Refresh* to finalize the config changes used in *GoWebScanProcess*.
5. Call *GoWebScanProcess_Start*.
6. Replay the detection, using the recording accessed via *GoWebScanCalCollector_GetCalRecording*.

Also, there is now a dedicated class for recording files: *GoWebScanRecording*. This is used in the *GoWebScanCalCollector*, and it has replaced the previously used recording class instances (kArrayLists) in both *GoWebScanProcess* (where recording can be enabled for detection mode), and in the example program (where a recording is loaded from disk to replay).

System Mode Switching

GoWebScanConfig now has functions to update the system mode and calibration without requiring restarting GoWebScan. The use case for this is mainly for running detection after calibration; the mode would be updated from calibration to detection mode, and the previously null calibration file would be updated with the newly created one. Before the config is updated, the process pipeline should be cleared with *GoWebScanProcess_Clear*. After the config has been updated, the process needs to be refreshed with a call to *GoWebScanProcess_Refresh*.

Outputting Calibration and Board Data Asynchronously

There is now functionality to process output data asynchronously, and the example program demonstrates how to use it. *GoWebScanProcess_SetDataHandler* may be used to set a callback function to process output data in a separate thread. This prevents any time-consuming processing functionality from blocking the input data thread. Similarly,

GoWebScanCalCollector_SetOnCompleteHandler may be used to set a callback function to process calibration data in a separate thread.

New Features for Calibration Replay

Calibration requires only slightly more data than necessary in order to proceed. This is not an issue during live calibration, but in the case of a calibration replay, the recording might only contain the exact amount of data that was used, and reprocessing might not trigger a calibration. To address this, replaying calibration should call *GoWebScanCalCollector_Flush* after all the data has been processed.

Replaying calibration recordings may also result in false frame drop warnings due to the recording containing a reduced amount of data. This can be avoided by disabling frame drops with the *GoWebScanCalCollector_CheckFrameDrops* function. The value set by this function can be read with *GoWebScanCalCollector_FrameDropsChecked*.

New Utility Functions

Utility functions have been created for saving, loading, and verifying the sensor info used alongside recordings. *GoWebScanUtils_StoreSensorInfo* can be used to store the sensor info at a specified path, *GoWebScanUtils_LoadSensorInfo* can be used to load the sensor info from a specified path, and *GoWebScanUtils_SensorInfoFilesExist* can be used to verify the sensor info files at a specified path.

GoWebScanUtils_AsyncSensorFunction can be used to asynchronously call a function on multiple GoSensor objects at once. Its usage is demonstrated in the example program to upgrade the firmware of all connected sensors simultaneously, by calling:

```
C/C++

...

kTest(GoWebScanUtils_AsyncSensorFunction(upgrade, context,
    (GoSensor*)kArrayList_Data(allSensors), kArrayList_Count(allSensors)));

...
```

where the upgrade function is as follows:

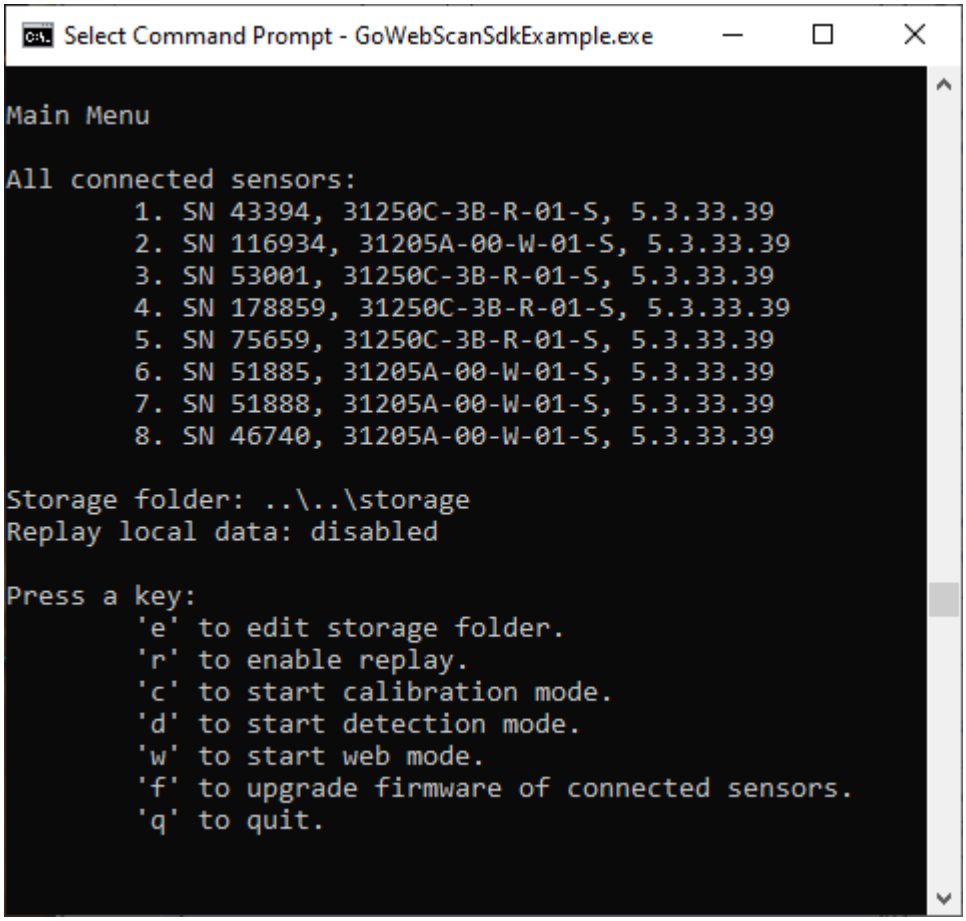
```C/C++
kStatus kCall upgrade(void* ctx, GoSensor sensor)
{
    DataContext* context = (DataContext*)ctx;

    printf("\tSN %d firmware upgrade started.\n", GoSensor_Id(sensor));

    if (kSuccess(GoSensor_Upgrade(sensor, context->firmwarePath, kNULL, kNULL)))
        printf("\tSN %d firmware upgrade complete.\n", GoSensor_Id(sensor));
    else
        printf("\tSN %d firmware upgrade failed.\n", GoSensor_Id(sensor));

    return kOK;
}
```

Resetting Calibration

GoWebScanCalCollector_Reset and *GoWebScanCalProcessor_Reset* have been added to reset the state of the collector and processor in order to rerun a live or replayed calibration without having to reopen the program.

Appendix: Changes to the Example Program

The program has been rewritten to display a more detailed menu at the start:

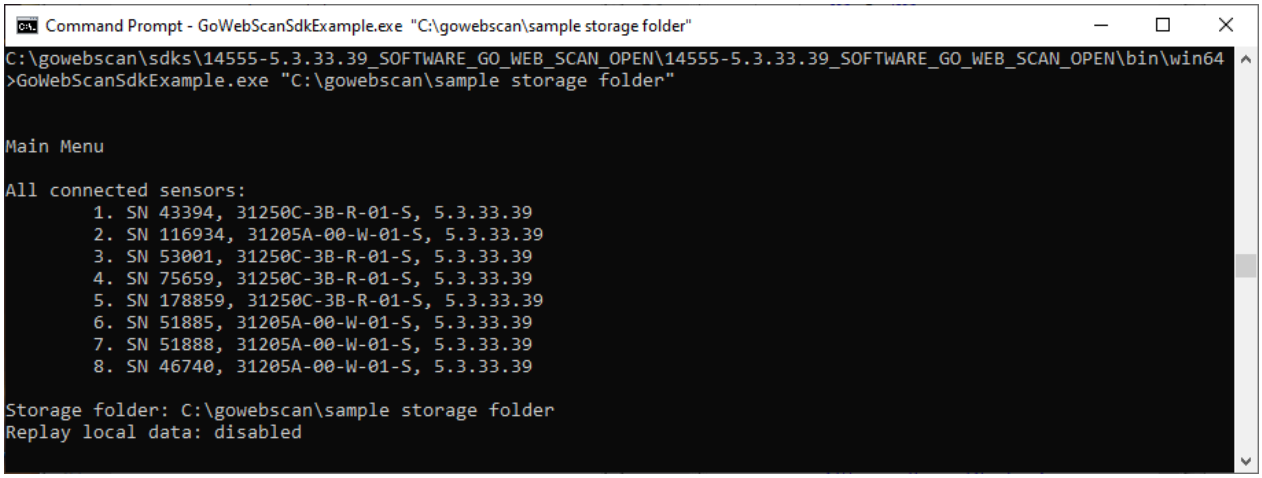


The menu includes a list of all the connected sensors, as well as their firmware versions, the path of the storage folder, and whether or not replay is enabled.

The main menu also includes options to edit the storage folder, toggle replay mode, enter calibration mode, enter detection mode, enter the newly-added web mode, upgrade the firmware of all connected sensors, or quit the program.

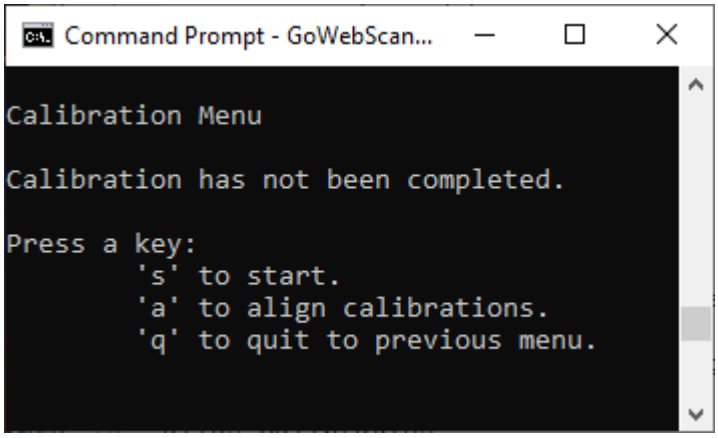
Specifying Storage Folder as an Argument

The storage folder can be edited from the main menu. It can also be chosen by specifying an optional argument when running the program:

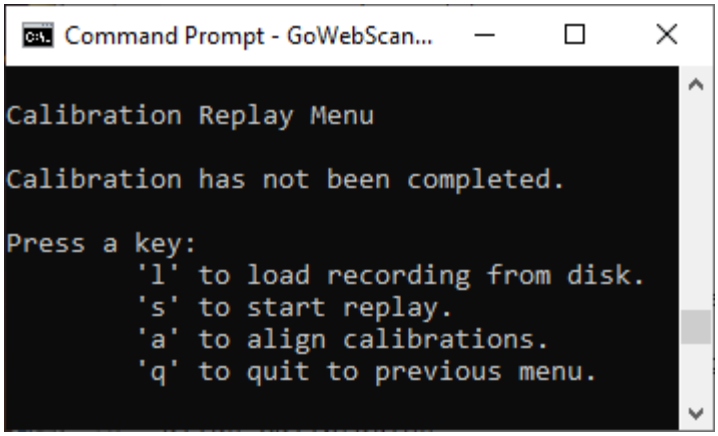


Calibration Menu

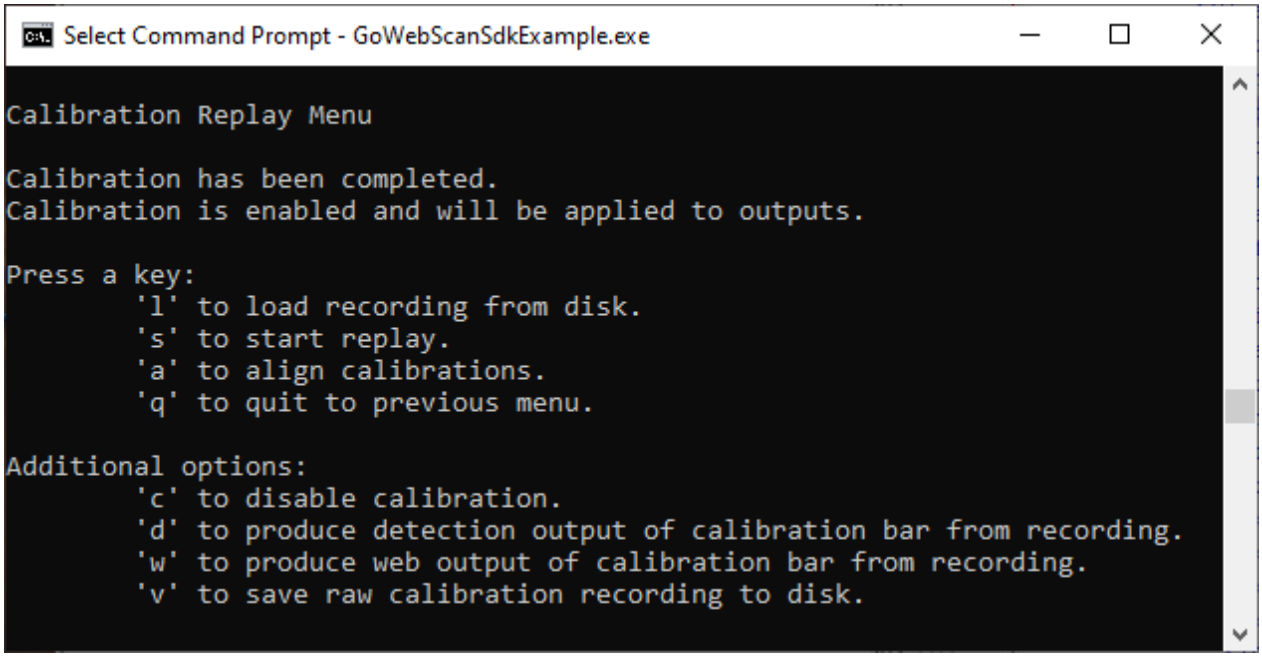
Selecting calibration mode from the main menu leads to options to start calibration, align calibrations of multiple systems, or quit to the main menu:



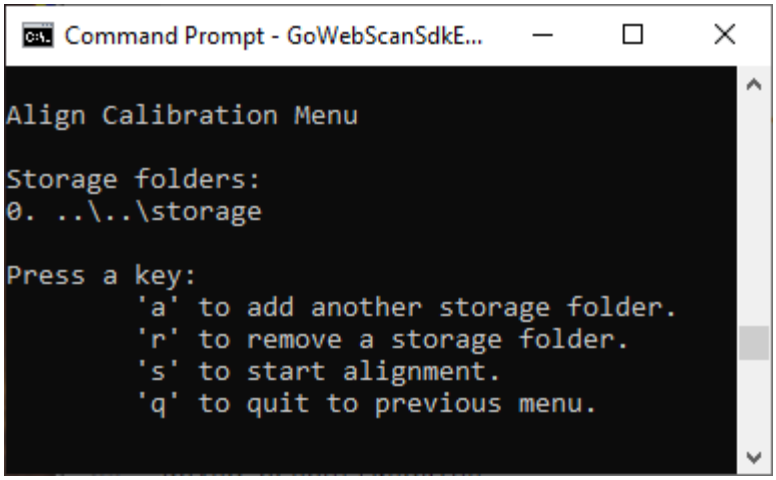
Calibration mode with replay enabled looks similar, but has options to load a recording and start replaying it rather than starting a calibration with live data:



Successfully completing a calibration, either by live data or replay unlocks new menu options, including toggling whether or not calibration is enabled for subsequent data produced, producing the detection output of the calibration bar, producing the web output of the calibration bar, or saving the calibration data as a recording to disk:

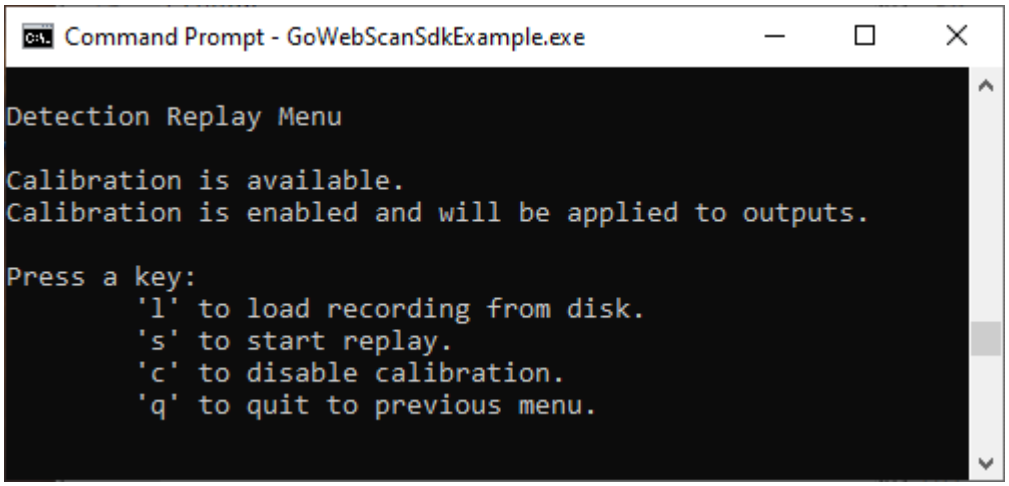


Entering the option to align calibration results in a list of storage folders representing the different systems to be aligned, and options to add to this list, remove from this list, start the alignment, or quit to the previous menu:



Detection and Web Replay Menu

Selecting detection or web mode from the main menu with replay enabled results in options load a recording, start replaying it, toggle whether or not calibration is used in the replay, or quit to the previous menu:



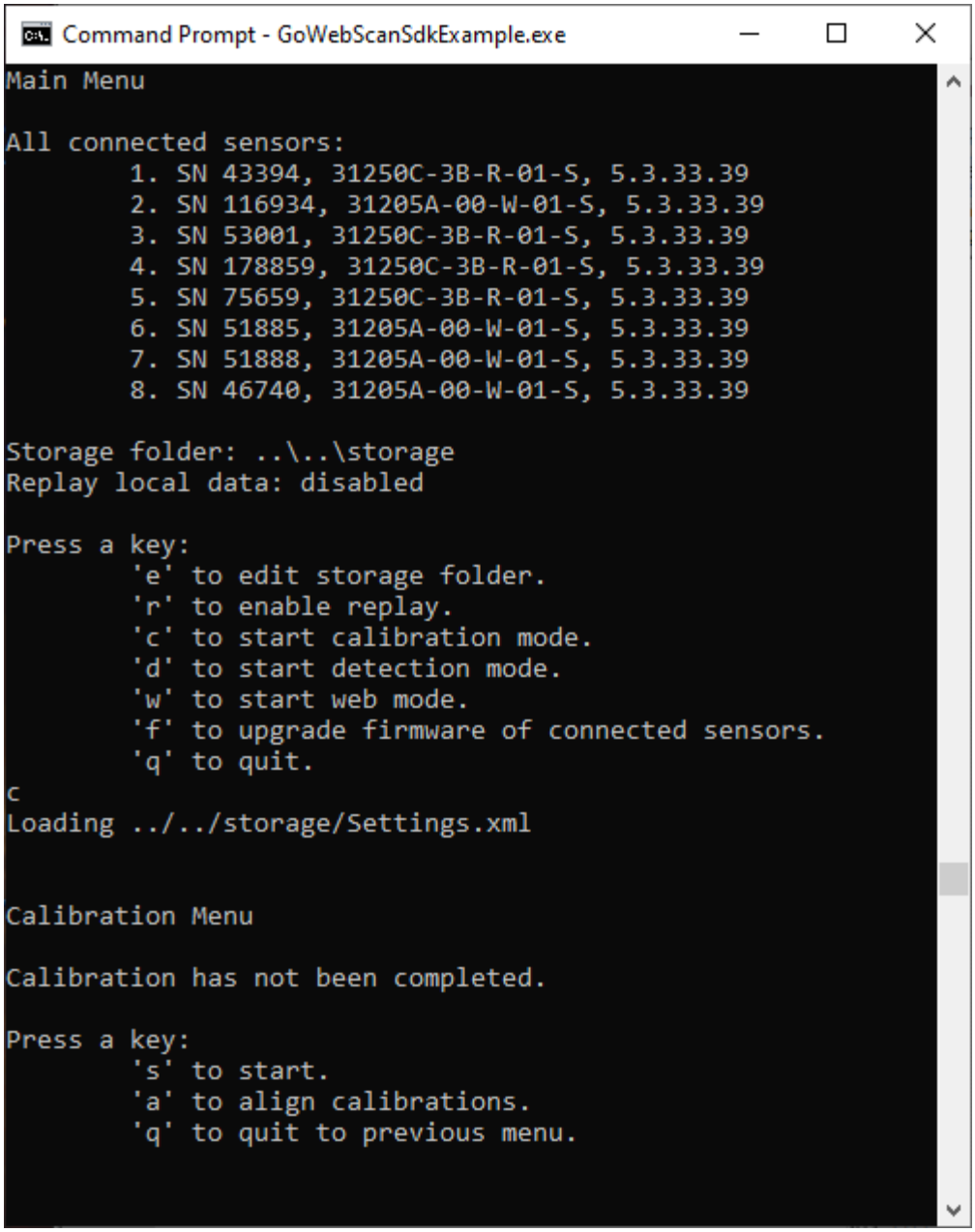
Appendix: Recording and Replaying Offline Web or Detection Data

In order to replay GoWebScan recordings without sensors connected, the GeoCal.xml files (one per sensor) and GoWebScanConfigSensorInfo.xml files (one per profile sensor) must be saved alongside the recordings.

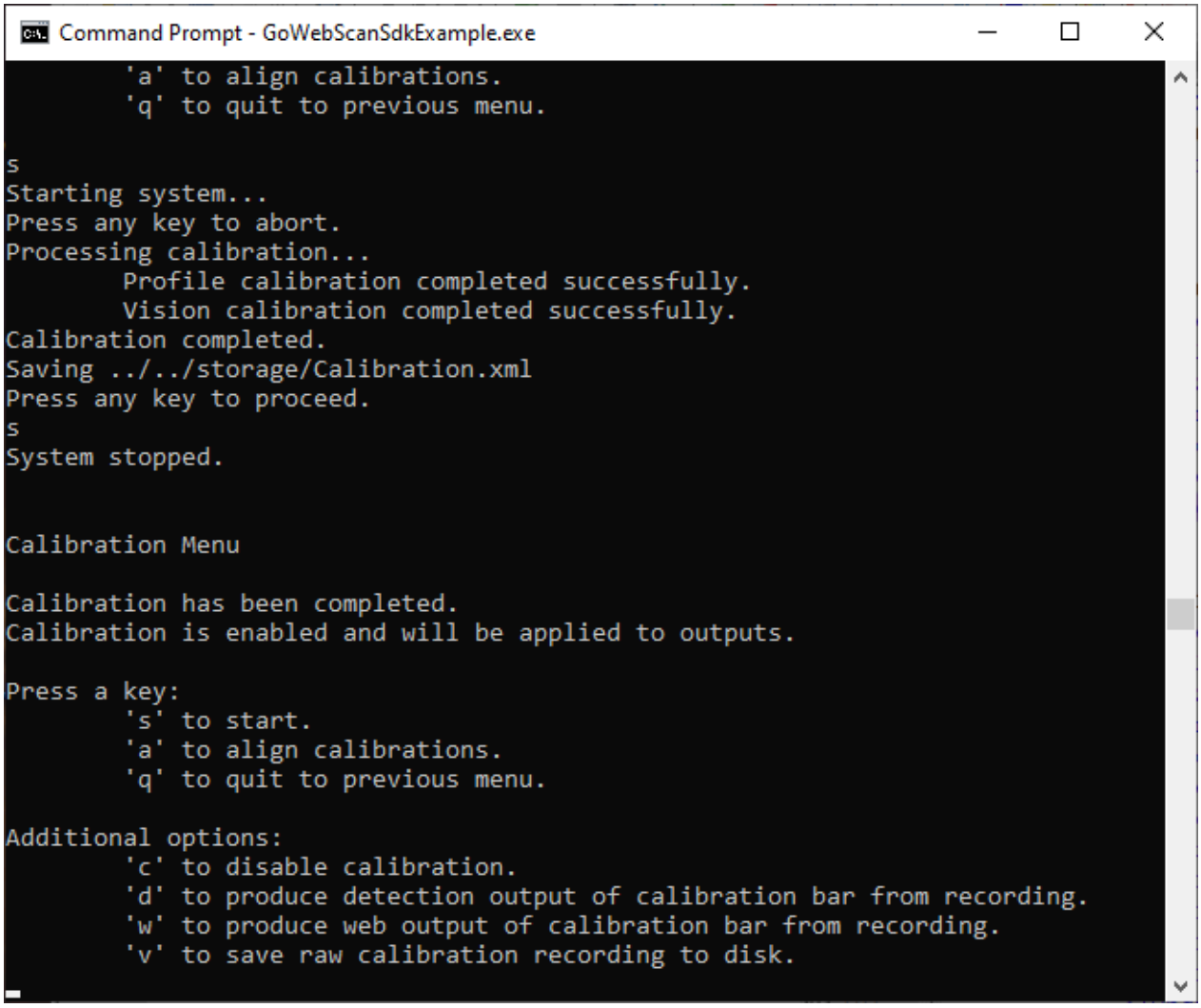
Recording Data in Calibration Mode

To save the recording and info files in calibration mode, edit the Settings.xml file as desired, run the GoWebScanSdkExample.exe application and perform the following steps:

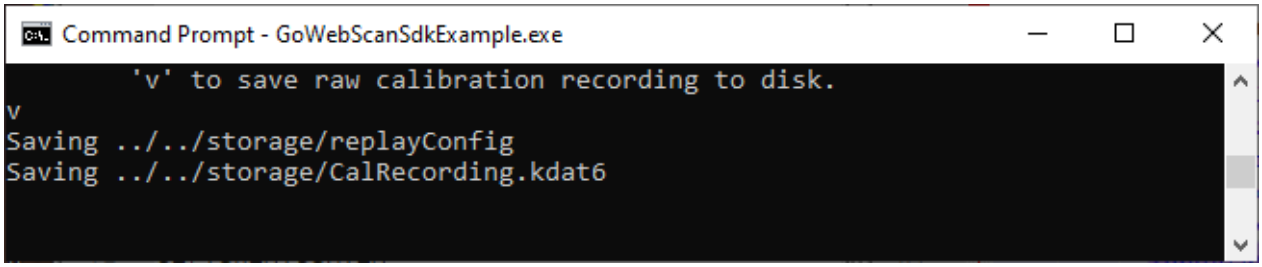
- From the main menu, enter 'c' for calibration mode and ensure that offline replay mode is **disabled**.



- Press 's' to start, then perform a calibration, as normally done.



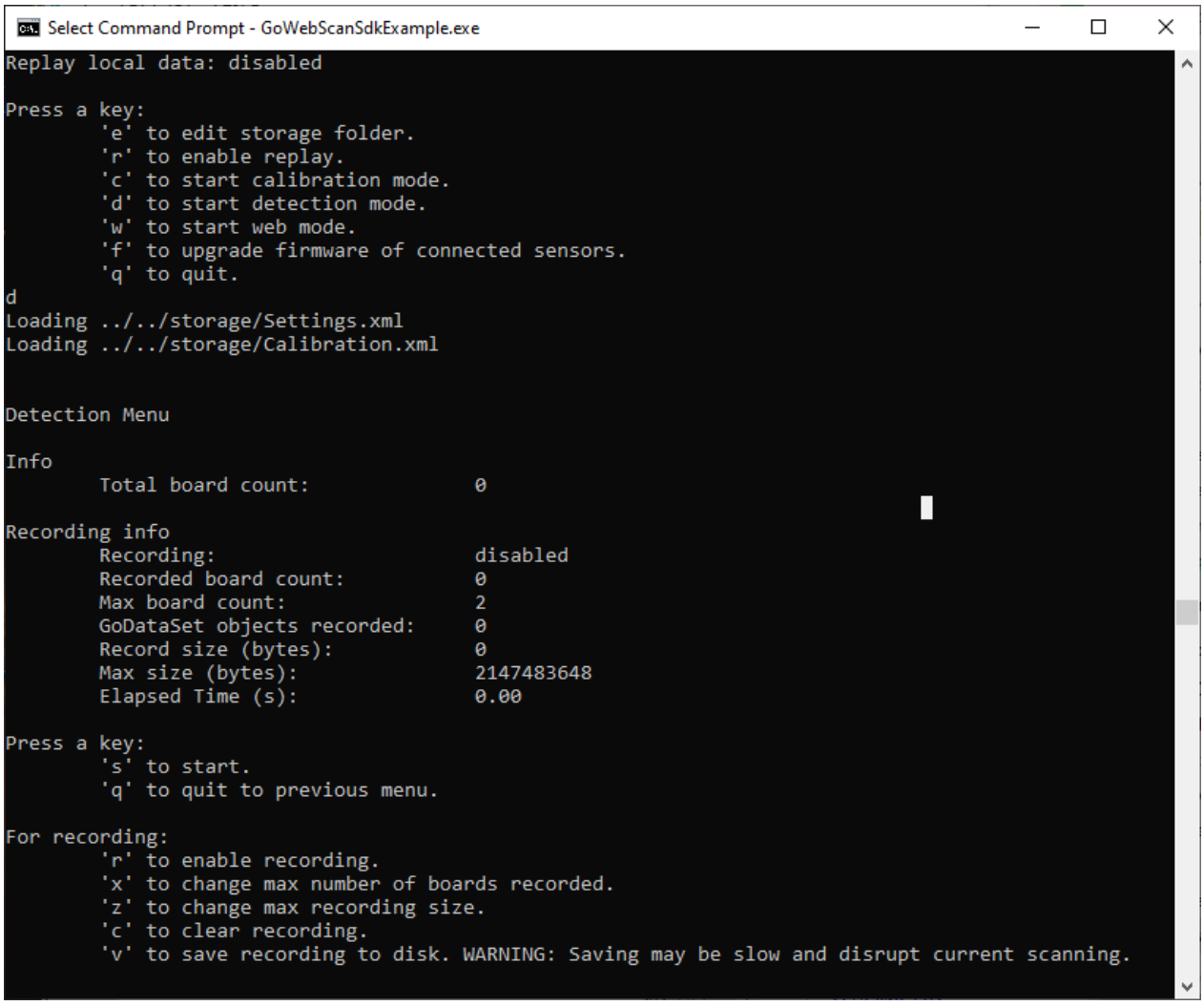
- Enter 'v' to save the calibration bar data recording.



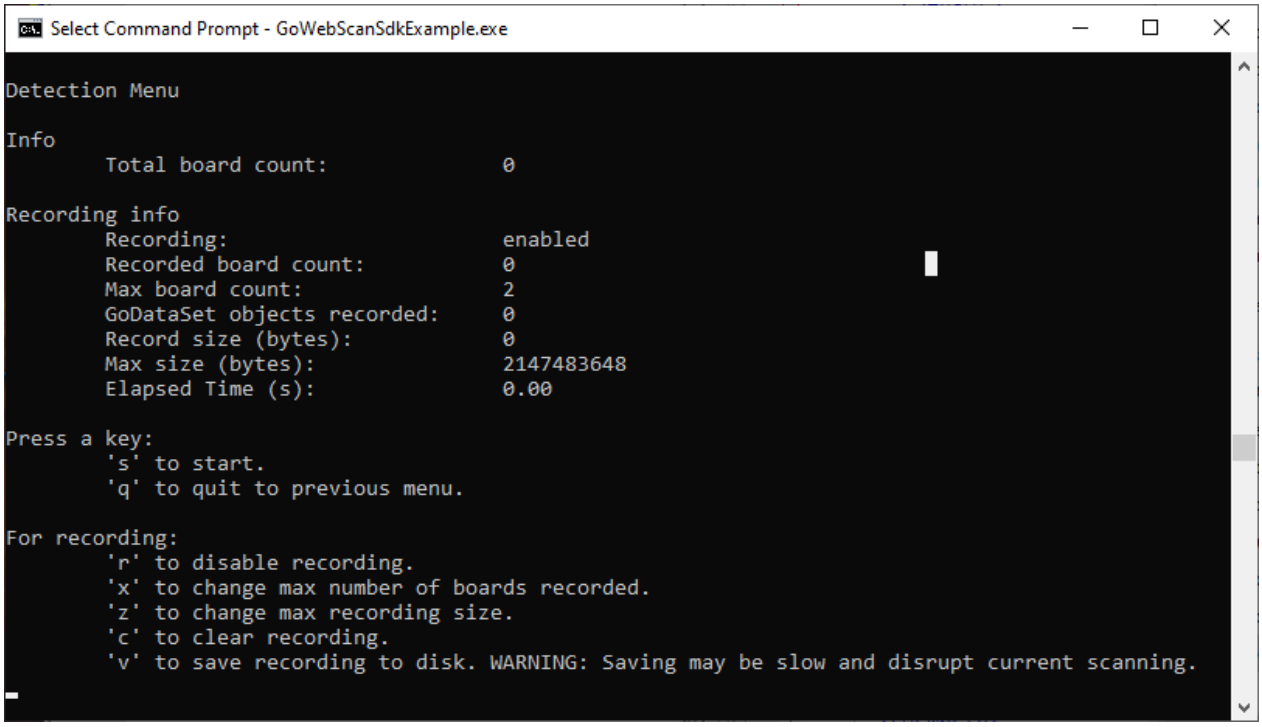
The storage folder should now contain the recording file ("CalRecording.kdat6", in this case), as well as a folder called **replayConfig** that contains the GeoCal.xml and GoWebScanConfigSensorInfo.xml info files.

Recording Data in Detection or Web Mode

- From the main menu, enter ‘d’ for detection mode or 'w' for web mode and ensure that offline replay mode is disabled.



- Enter 'r' to enable recording.



Recall that for large recordings, the recording size may need to be increased. Also, if several boards are going to be recorded, the max board count would need to be increased.

Once the desired setup is chosen:

- Press 's' to start, then scan board data, as normally done.

Once the data is recorded:

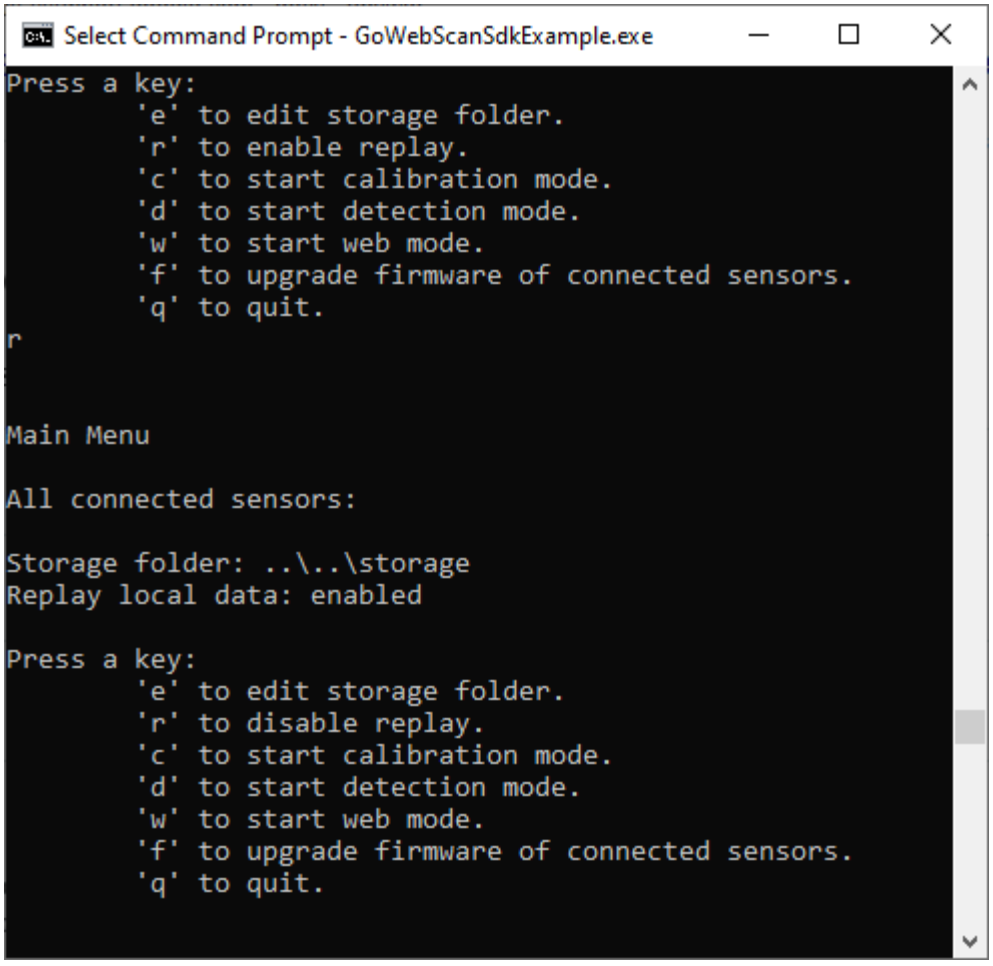
- Press any key to stop.
- Press 'v' to save the recording to disk, along with the sensor info files.

The storage folder should now contain the recording file ("Recording.kdat6", in this case), as well as a folder called **replayConfig** that contains the GeoCal.xml and GoWebScanConfigSensorInfo.xml info files.

Replaying Data

To replay the recording, the Settings.xml file, .kdat6 recording file and replayConfig folder and its contents are necessary (if the recording file or replayConfig folder is missing, first follow the above procedure to record data). Disconnect the sensors from the computer, run the GoWebScanSdkExample.exe application and perform the following steps:

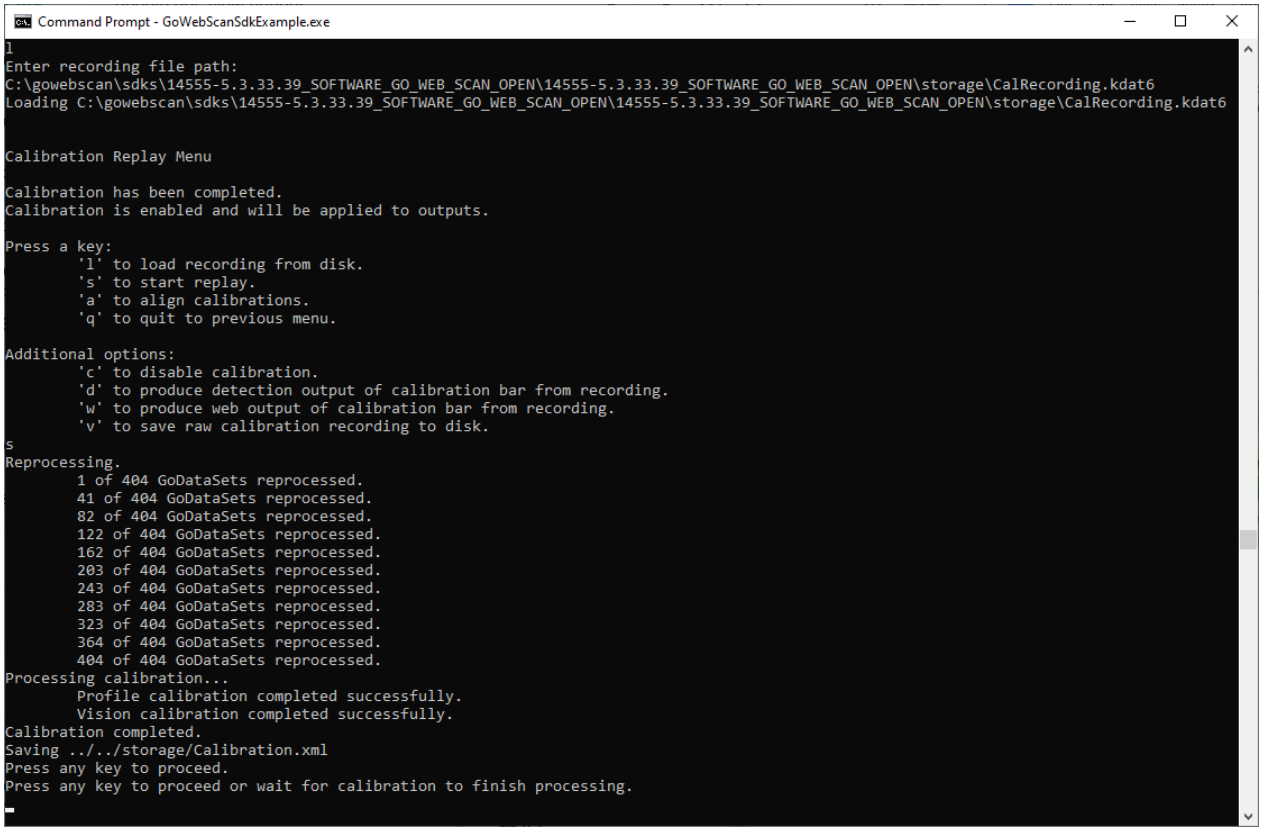
- Enter 'r' to enable replay.



- Enter the desired mode (‘c’ for calibration, ‘d’ for detection, or ‘w’ for web). Doing so will load the sensor info.

Replaying Data in Calibration Mode

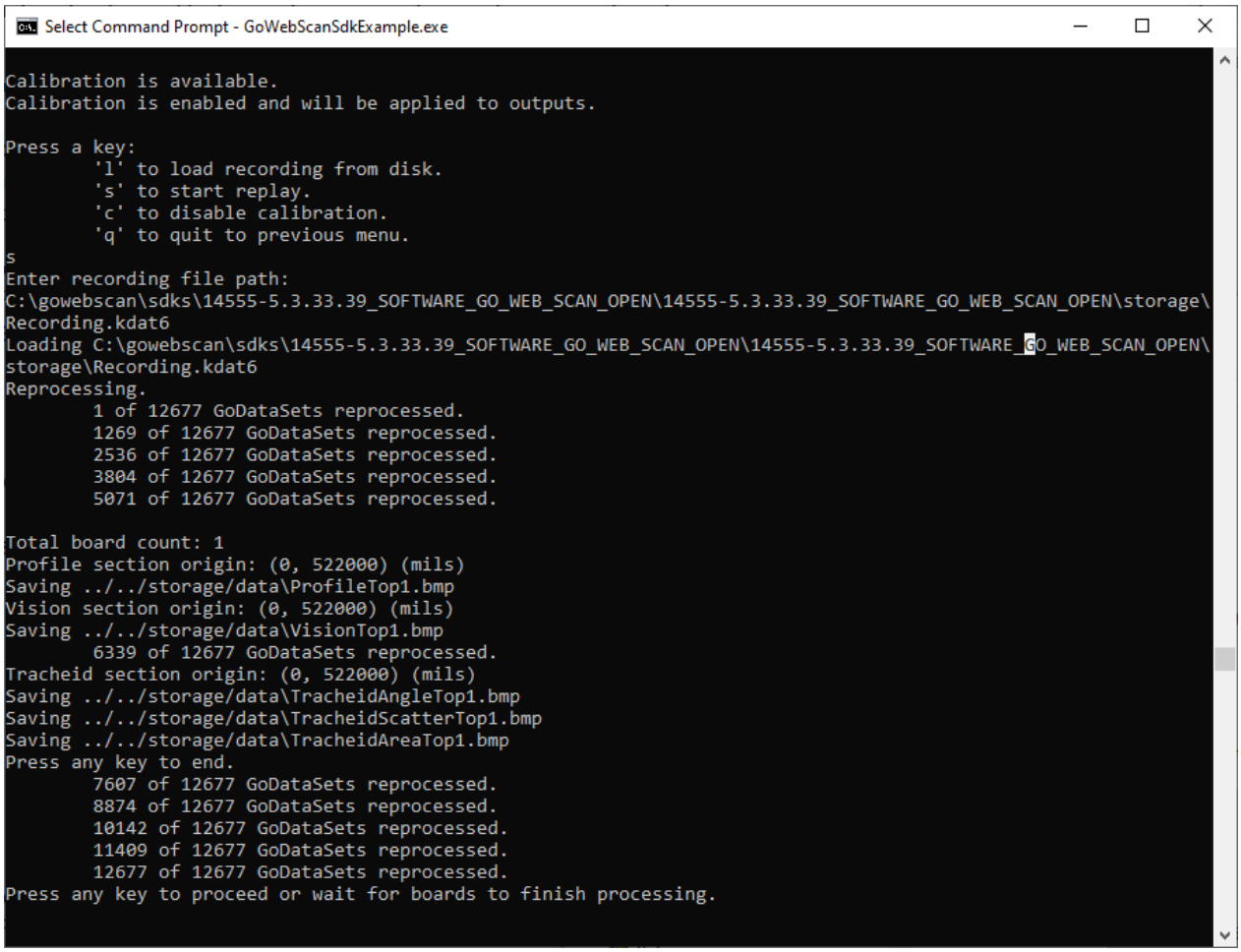
To replay data in calibration mode, press 'l' to load a recording then enter the recording file path, then press 's' to start. Starting without a loaded recording will result in being prompted for one.



The program should then reprocess the recording and produce the exact same calibration that was created when the recording was originally made.

Replaying Data in Detection or Web Mode

To replay data in detection or web mode, in the detection or web menu press 'l' to load a recording then enter the recording file path, then press 's' to start. Starting without a loaded recording will result in being prompted for one.



The program should then reprocess the recording and produce the exact same output that was created when the recording was originally made (unless the recording was done in a different mode than the replay).

Note that the function *GoWebScanProcess_Clear* is called before each attempted replay in the code. It is necessary to call this to clear the system in the event that a recording to be processed contains earlier encoder value data than what was processed more recently.