# Intel® RealSense™ Depth Module D400 Series Custom Calibration

*Revision 1.0.0*

*January 2018*

# Contents

# Tables

# List of Figures

# Revision History

| Document Number | Revision Number | Description | Revision Date |
|---|---|---|---|
| XXXXXX | 1.0 | Initial Release | 01/2018 |
| | | | |

# 1 Introduction

## 1.1 Purpose and Scope of This Document

In order to operate Intel® RealSense™ D400 device efficiently and accurately, user needs to make sure the device is well calibrated. The Intel supplied calibration tools including Intel® RealSense™ Dynamic Calibrator and OEM Calibration Tool for Intel® RealSense™ Technology are designed to calibrate the devices with Intel proprietary algorithms. In certain cases, user desires the ability to use their own algorithms and still be able to calibrate the device with optimized parameters from the custom algorithm.

This document contains technical information to assist those developing custom calibration solutions for Intel® RealSense™ D400 series modules. The primary goal is to guide the user to understand the process and facilities for the necessary configuration for the device under calibration, the definition of the parameters, and interfaces to update the parameters into the device. A simple but complete sample application with currently available OpenCV algo is used as an example. It's not in the scope of this document to discuss details of calibration algorithm or accuracy.

Developing custom calibration requires knowledge in computer vision as well as good understanding of the RealSense device operating details. It is a complex topic and intended only for those with expert level knowledge.

## 1.2 Organization

This document is organized into four main parts: overview, setup, calibrating a device with custom calibration sample app, and developing custom solution:

- **Overview** – brief overview of the Calibration API and parameters.
- **Setup** – hardware and software setup for running the Custom Calibration Sample App to calibrate a device and developing custom calibration solutions.
- **Calibrating Device with Custom Calibration Sample App** – describes the necessary hardware and software setup required running the Custom Calibration Sample App and details steps to calibrate device.
- **Developing Custom Calibration Solution** – uses the sample application as an example to describe details of steps implementing a custom calibration solution for Intel® RealSense™ D400 series modules.

# 2 Overview

## 2.1 Dynamic Calibration API and Calibration Data Read/Write

Intel provides a software interface in Dynamic Calibration API to enable user uploading those calibration parameters to Intel® RealSense™ D400 devices and read the parameter back from device:

- WriteCustomCalibrationParameters – write parameters to device
- ReadCalibrationParameters – read parameters from device

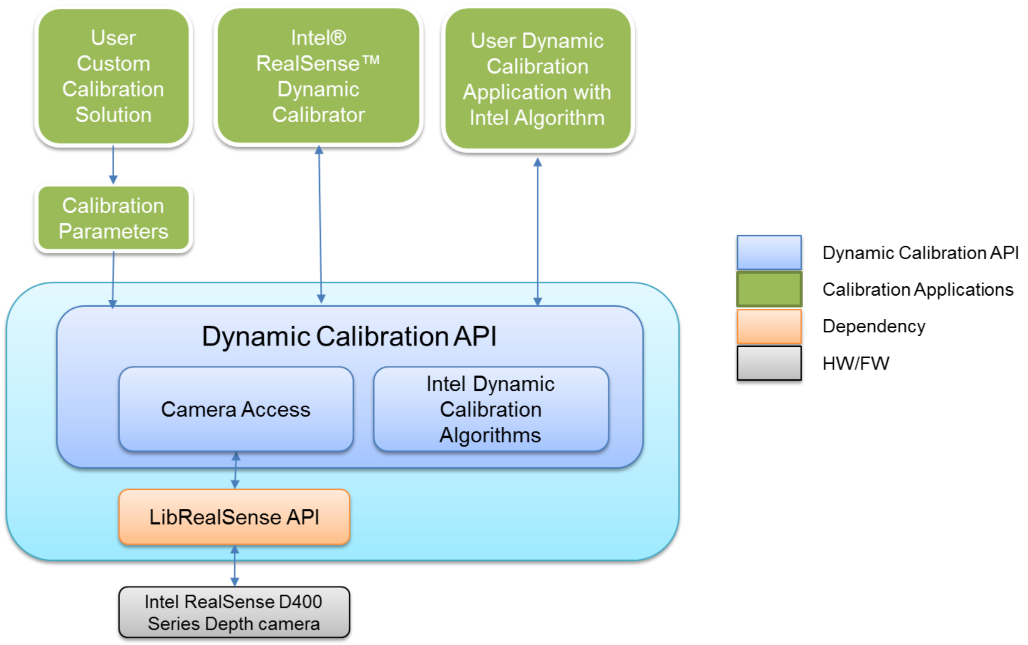

User Custom Calibration Solution
Intel® RealSense™ Dynamic Calibrator
User Dynamic Calibration Application with Intel Algorithm
Calibration Parameters
Dynamic Calibration API
Camera Access
Intel Dynamic Calibration Algorithms
LibRealSense API
Intel RealSense D400 Series Depth camera
Dynamic Calibration API
Calibration Applications
Dependency
HW/FW

**Figure 2-1 Software Stack with Dynamic Calibration API and Calibration Apps**

An example tool CustomRW is also included in Dynamic Calibration API to read calibration parameters from XML file and write them to the device.

A user custom calibration app can choose one of the two approaches to update the results to the device:

- To link to the WriteCustomCalibrationParameters and ReadCalibrationParameters and write directly to the device through the APIs.

- To write the results into a parameter XML file and then use CustomRW to read/write the parameters to the device.

## 2.2 Calibration Parameters

Calibration parameters includes INTRINSICS and EXTRINSICS. Assume left camera is the reference camera and is located at world origin. RGB parameters only apply to modules with RGB, e.g., D415 and D435.

Intrinsic includes

- Focal length - specified as [fx; fy] in pixels for left, right, and RGB cameras
- Principal point - specified as [px; py] in pixels for left, right, and RGB cameras
- Distortion - specified as Brown's distortion model [k1; k2; p1; p2; k3] for left, right, and RGB cameras

Extrinsic includes

- RotationLeftRight - rotation from right camera coordinate system to left camera coordinate system, specified as a 3x3 rotation matrix
- TranslationLeftRight - translation from right camera coordinate system to left camera coordinate system, specified as a 3x1 vector in millimeters
- RotationLeftRGB - rotation from RGB camera coordinate system to left camera coordinate system, specified as a 3x3 rotation matrix
- TranslationLeftRGB - translation from RGB camera coordinate system to left camera coordinate system, specified as a 3x1 vector in millimeters

The calibration data read/write API allows user to upload both INTRINSICS and EXTRINSICS. The user custom algorithm is free to optimize all these parameters. The sample in this document optimizes both intrinsic and extrinsic parameters.

The following table shows how the various calibration tools would impact calibration parameters for Intel® RealSense™ D400 series depth cameras.

| | Factory Calibration | OEM Calibration | Technician Calibration | User Custom Calibration | Dynamic Calibration |
|---|---|---|---|---|---|
| Intrinsic | x | x | x | X | |
| Extrinsic | x | x | x | x | x |

## 2.3 Impacts on Calibration Tables

Intel® RealSense™ D400 devices maintain three layers of internal calibration tables: Active (Dynamic) tables on flash, Gold tables on flash, and Factory tables on EEPROM:

- Each layer of tables include Coefficient Table, Depth Table, and RGB table where RGB table only exists on D415 and D435.
- D435 devices do not have RGB table on EEPRPOM (factory calibration did not write it because RGB is adds-on).

Custom Calibration updates the results to Active tables in flash on ASIC. It does not touch the tables in EEPROM which is on the optics module and Gold tables on flash.

| | Factory Calibration | OEM Calibration | Technician Calibration | Custom Calibration | Dynamic Calibration |
|---|---|---|---|---|---|
| Active Tables in Flash | x | x | x | x | x |
| Gold Tables in Flash | x | x | x | | |
| EEPROM | x | | | | |

## 2.4 Frame Formats Used in Custom Calibration

The following unrectified calibration frame formats are available. The Custom Calibration Sample App uses these formats.

**Table 2-1. Frame Formats Used in Custom Calibration**

| Format | SKU | Used | Comment |
|---|---|---|---|
| Y16 (16-bit) | D400 | Left and Right Sensors: 1920x1080 @ 15 FPS | Intel® RealSense™ Camera D400, D410, D415 |
| | D410 | | |
| | D420 | Left and Right Sensors: 1280x800 @ 15 FPS | Intel® RealSense™ Camera D430 D420, D435 |
| | D430 | | |
| YUY2 | D415 | RGB Sensor: 1920x1080 @ 15 FPS | Intel® RealSense™ Camera D415, D435 |
| | D435 | | |

## 2.5 Frame Sync

The calibration frames are not synced. To avoid possible motion blur, calibrate the device with images captured at static positions. In this sample, a tripod is used to keep the camera device at fixed positions while capturing the images for calibration. This is same if you want to develop your own custom calibration app.

## 2.6 Accuracy

The calibration will not be accurate if the checkerboard doesn't sufficiently cover the field of view or the poses are not diverse enough between captures. The calibration will also not be accurate if the camera or board is handheld (or not completely still) as there is no framesync in calibration mode.

# 3 Setup

This section describes the required hardware and software setup for running the Custom Calibration Sample Application to calibrate a device and developing a Custom Calibration Application.

## 3.1 Hardware

The hardware required including a calibration target, D400 series device to be calibrated, USB cable, and a Windows 10 computer. To ensure the images are synced, pictures are captured at static device positions, a tripod is used to support the device during calibration.

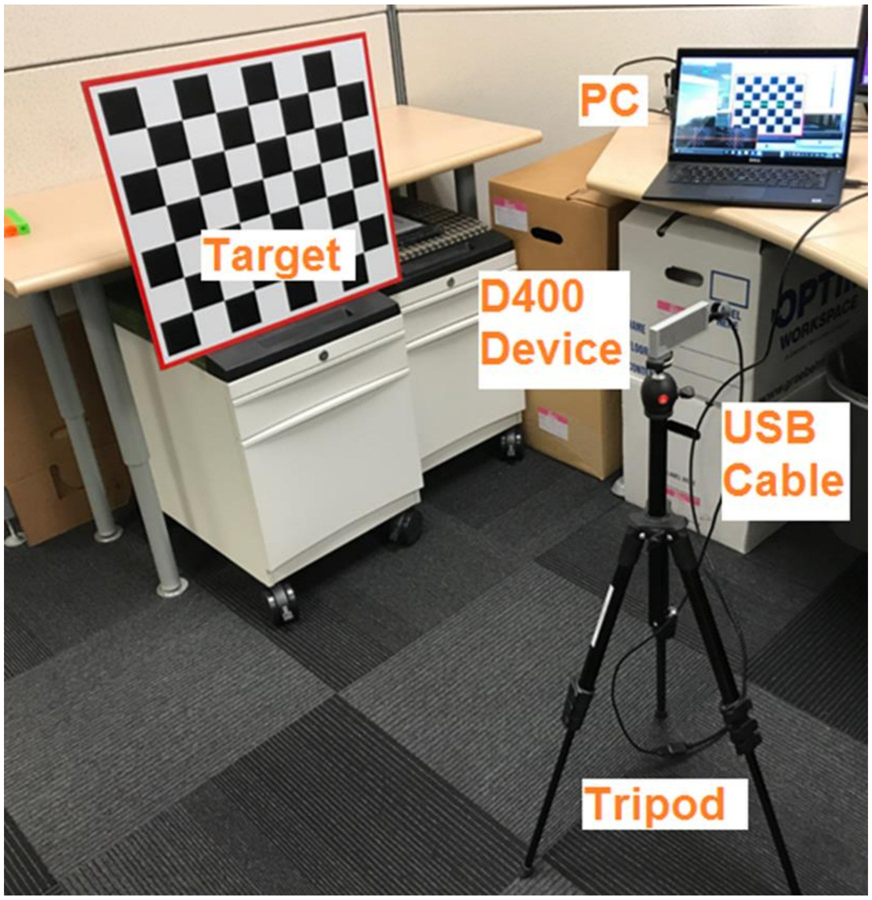


**Figure 3-1 Hardware Setup**

### 3.1.1 Device

Intel® RealSense™ D415 device as shown below is used to show case the custom calibration process. All D400 series devices can use the same process.

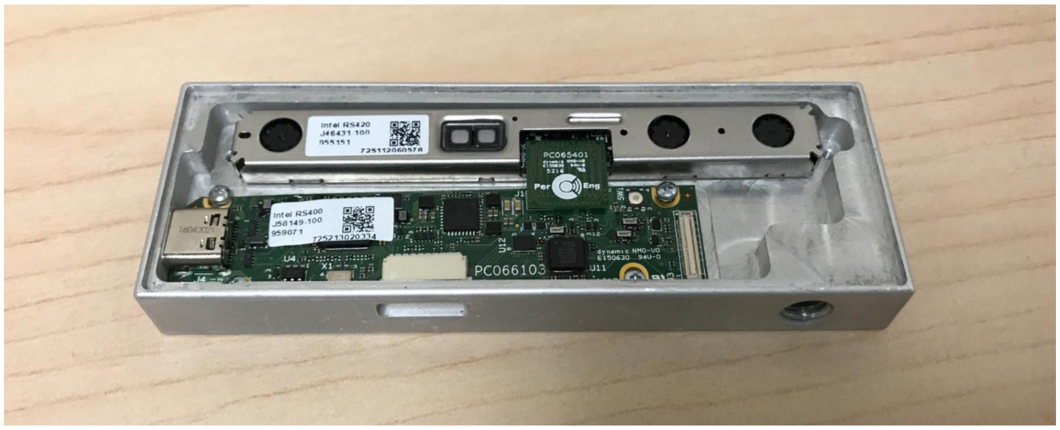


**Figure 3-2 D415 Device**

### 3.1.2 Target

The target used in this custom calibration example is an 8x7 checkerboard with a 60x60mm checker size. The target image pdf 540x480_60mm.pdf is included in the software package described in *3.2.1 Custom Calibration Sample Application.* The target pdf is located under the target directory. Users must ensure that when printing the target, the target is not scaled.

A developer may choose to use a different target, but will need to modify the sample application accordingly and recompile.

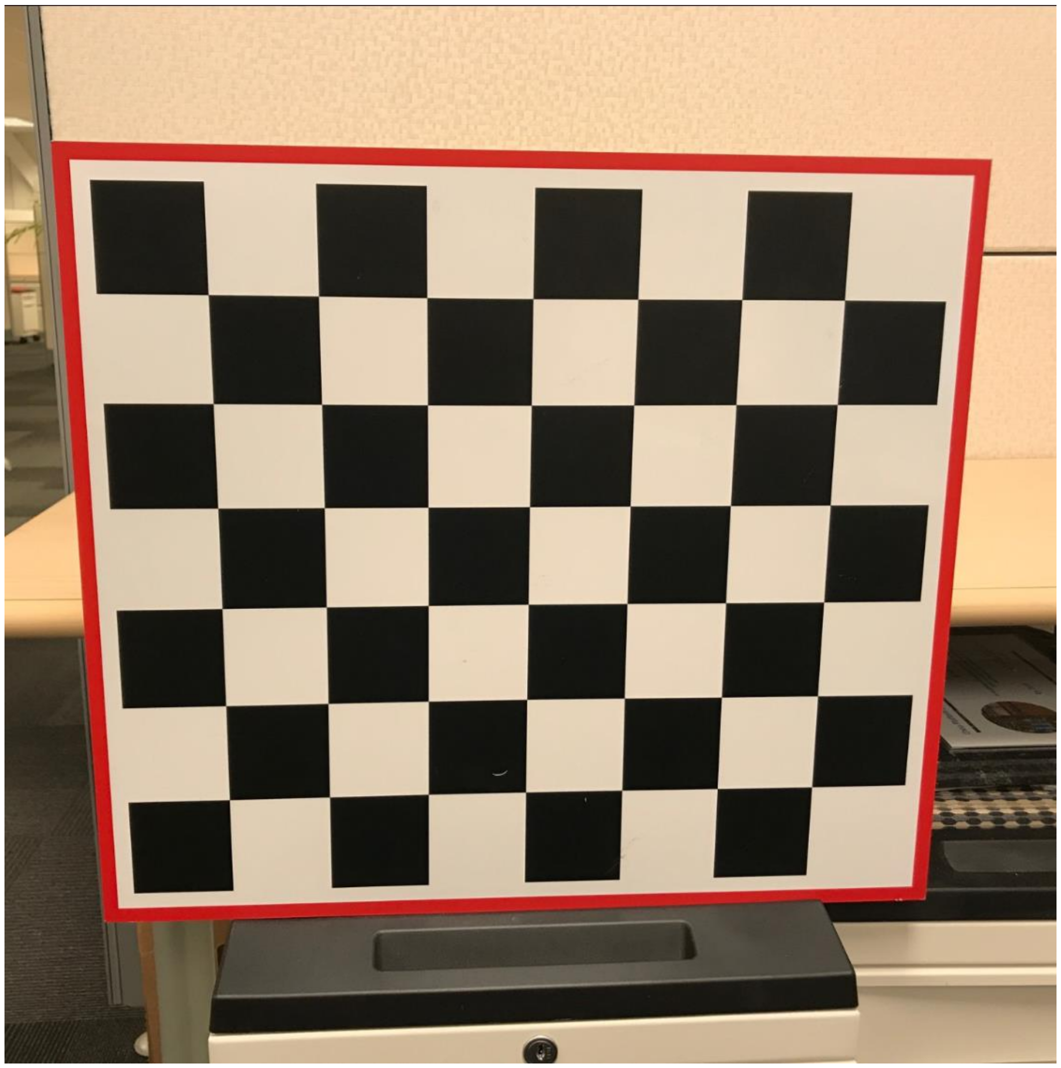


Figure 3-3 8x7 60x60 mm Checker Calibration Target

### 3.1.3 Tripod

Any medium sized tripod should be sufficient. In this example setup, we used a Manfrotto Compact Light Aluminum Tripod with ball head. The ball head makes adjustment to device orientation easier.

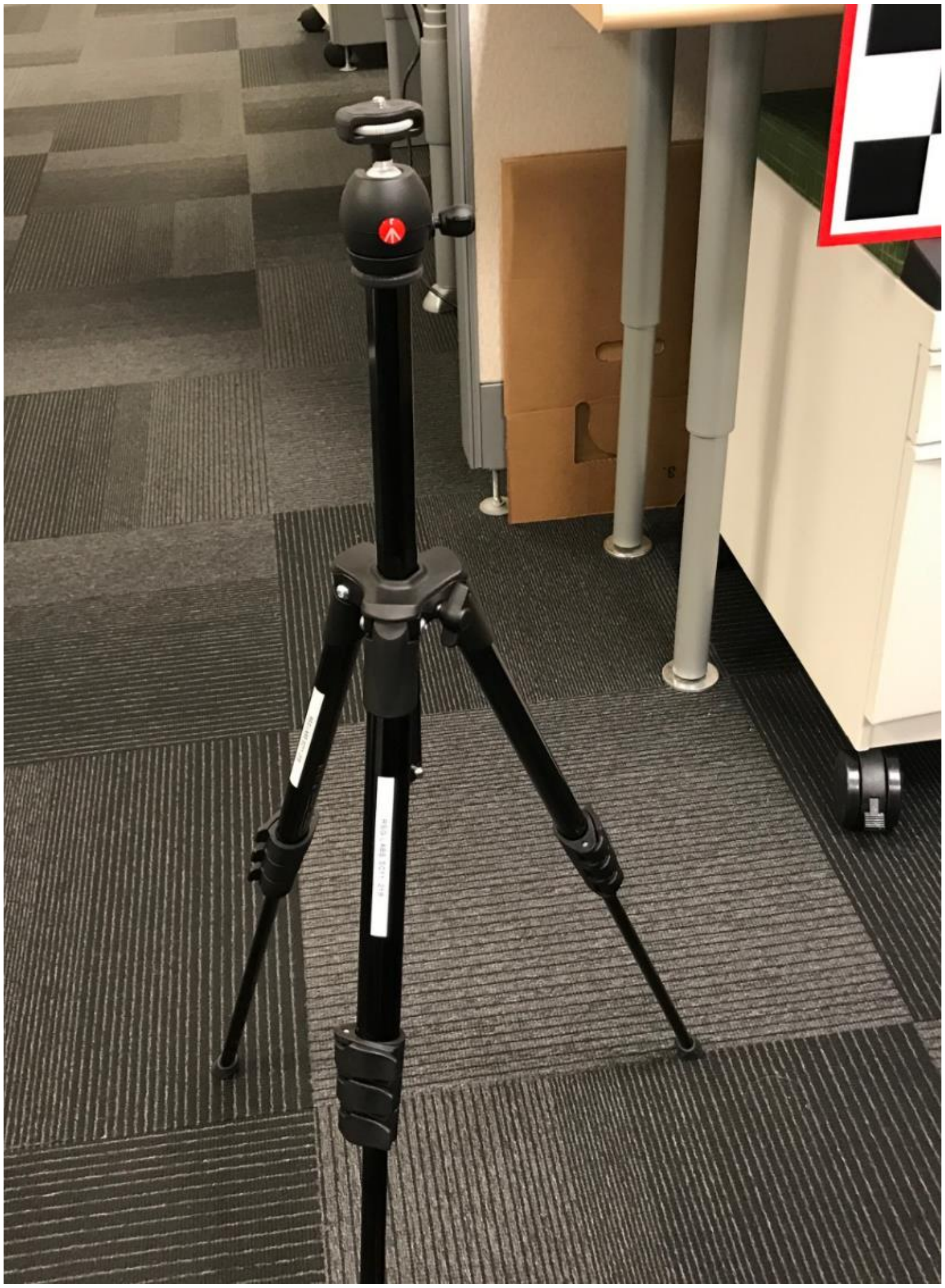


Figure 3-4 Tripod

### 3.1.4 USB

A long USB type C cable to connect the device to the host computer where the custom calibration sample application will run.

### 3.1.5 PC

A Windows 10 computer.

## 3.2 Software

Install the following to the windows computer.

### 3.2.1 Custom Calibration Sample Application

A CustomCalibration.zip package is provided for the Custom Calibration Sample Application. It includes both binary executables and complete source code. Unzip the package:

```
CustomCalibration
       |
       |--------- bin
       |              |-------- CustomCalibrationTest.exe
       |
       |--------- CustomResult
       |
       |--------- Include
       |              |------- GL
       |              |------- libpng
       |
       |--------- Lib
       |              |------- freeglut
       |              |------- libpng
       |
       |--------- librealsense
       |               |------- include
       |               |------- lib
       |
       |--------- OpenCV
       |--------- Sample
       |               |------- CustomCalibrationTest.sln
       |               |…
       |
       |--------- Target
       |               |------- 540x480_60mm.pdf
```

**Bin** folder contains the executables and dependent runtime libraries. CustomCalibrationTest.exe is the executable for the Custom Calibration Sample Application.

**CustomResult** contains all device calibration results**.**

**Target** contains the target image pdf 540x480_60mm.pdf. User will need to print it without scaling.

**Sample** contains the full source code of the Custom Calibration Sample Application.

**Include**, **Lib**, **librealsense**, and **OpenCV** are dependent libraries and headers in order to compile Custom Calibration Sample Application. Librealsense supports streaming from the device. OpenCV supplies the calibration algo.

### 3.2.2 Intel® RealSense™ Dynamic Calibration API

The calibration example uses support functions provided by the Intel® RealSense™ Dynamic Calibration API to store updated calibration parameters into the non-volatile storage on the Intel® RealSense™ D400 series modules. Required functions were first introduced in version 2.5.0.0 of the Dynamic Calibration API, so install the latest version and ensure that it is version 2.5.0.0 or later. ***Table 3-1*** lists where to download the Intel® RealSense™ Dynamic Calibration API software and associated documentation.

**Table 3-1. Intel® RealSense™ Dynamic Calibration API Resources**

| Resource | URL |
|---|---|
| **Intel® RealSense™ Depth Module D400 Series Dynamic Calibration Software Download** | https://downloadcenter.intel.com/download/27415/?v=t |
| **Intel® RealSense™ Depth Module D400 Series Dynamic Calibration User Guide** | https://cdrd.intel.com/v1/dl/getContent/574999.htm |
| **Intel® RealSense™ Depth Module D400 Series Dynamic Calibration Programmer Guide** | https://www.intel.com/content/www/us/en/support/articles/000026724.html |

Find Dynamic Calibration API installer and follow instruction to install. The latest version 2.5.1.0. After installation, the directory structure should look like below:

```
DynamicCalibrationAPI
                |
                |-------- 2.5.1.0
                |              |-------- bin
                |              |           |--------DynamicCalibrator.exe
                |              |           |--------DynamicCalibratorCLI.exe
                |              |           |-------- CustomRW.exe
                |
                |              |-------- Include
                |              |-------- attributions.txt
                |              |-------- release_notes.txt
                |              |-------- license.txt
                |              |-------- safeclib
                |              |-------- gimbal
                |              |-------- examples
                |              |-------- target
```

CustomRW.exe under bin directory is the calibration data read/write tool. We will be using it to update the calibration parameters to the device after running the Custom Calibration Sample App and obtained optimized parameters for the device.

### 3.2.3 Intel® RealSense™ SDK

Install the latest release of the Intel® RealSense™ SDK so your custom calibration application can use LibRealSense to conveniently open and access Intel® RealSense™ D400 series modules. ***Table 3-2*** contains pointers to the SDK homepage, GitHub* repository where you can download the latest release, and the SDK documentation.

**Table 3-2. Intel® RealSense™ SDK Resources**

| Resource | URL |
|---|---|
| **Intel® RealSense™ SDK Home Page** | https://software.intel.com/en-us/realsense/sdk |
| **LibRealSense GitHub*** | https://github.com/IntelRealSense/librealsense |
| **SDK Documentation** | https://github.com/IntelRealSense/librealsense/tree/master/doc |

### 3.2.4 OpenCV 3.3.0

The calibration example presented in this document uses several OpenCV 3.3.0 libraries for computations, so OpenCV 3.3.0 must be installed. ***Table 3-3*** lists URLs for downloading OpenCV 3.3.0, and ***Table 3-4*** lists the specific OpenCV libraries that are required by the example code.

**Table 3-3. OpenCV 3.3.0 Resources**

| Resource | URL |
|---|---|
| **OpenCV GitHub* Repository** | https://github.com/opencv/opencv |
| **OpenCV 3.3.0 Download** | https://github.com/opencv/opencv/releases/tag/3.3.0 |

**Table 3-4. OpenCV 3.3.0 Libraries Required for the Example**

| OpenCV 3.3.0 Library |
|---|
| ittnotify.lib |
| opencv_calib3d330.lib |
| opencv_core330.lib |
| opencv_features2d330.lib |
| opencv_flann330.lib |
| opencv_imgproc330.lib |
| zlib.lib |

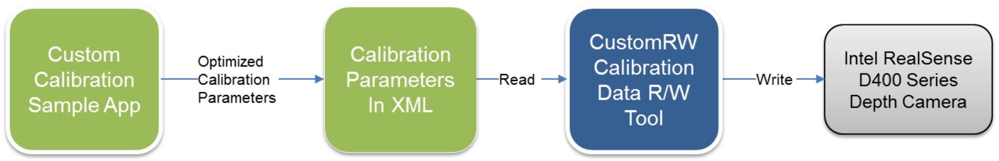
### 3.2.5 Glut Library

The FreeGLUT package is a convenient open source alternative to the OpenGL Utility Toolkit for displaying graphical output to the operator of the custom calibration application. You can download and install the FreeGLUT package at http://freeglut.sourceforge.net/.

# 4 Calibrating Device with Custom Calibration Sample Application

## 4.1 Process Overview

The general process to calibrate a device with the Custom Calibration Sample App involves running the app capturing images of the target from various viewpoints, the app optimizes the calibration parameters based on the captured images and writes the results to a XML file. The user then run CustomRW calibration data R/W tool to read the parameters from the XML file and write them to the device.


Custom Calibration Sample App
Optimized Calibration Parameters
Calibration Parameters In XML
Read
CustomRW Calibration Data R/W Tool
Write
Intel RealSense D400 Series Depth Camera

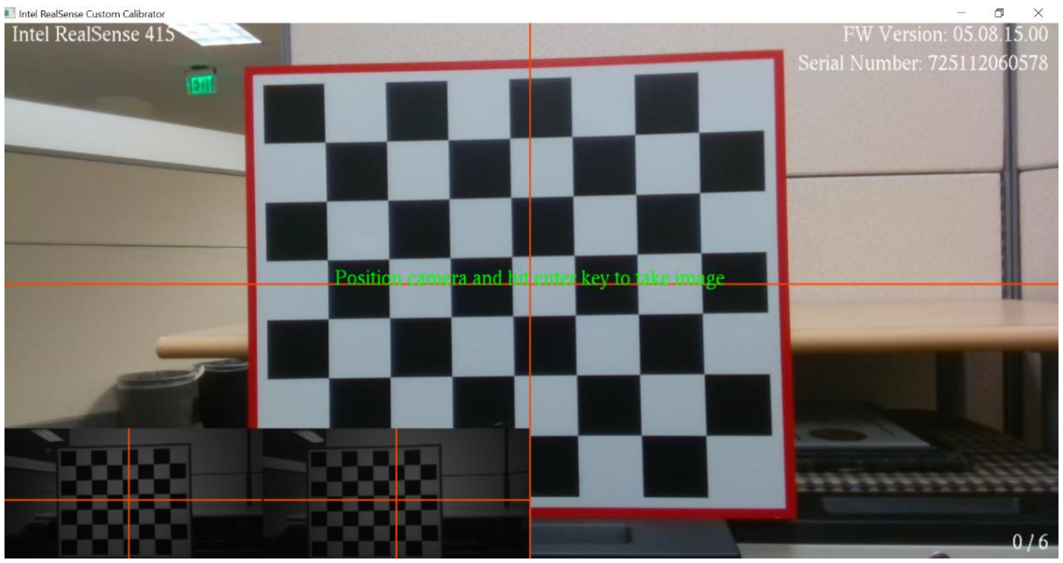
## 4.2 Connect Device to Computer

Connect the device through the USC cable to the PC where Custom Calibration Sample App locates.

## 4.3 Running Custom Calibration Sample Application

### 4.3.1 Starting Application

Use Windows explorer to browse to CustomCalibration\bin directory and find the CustomCalibrationTest.exe and double click it to start running.


Intel RealSense Custom Calibrator
Intel RealSense 415
FW Version: 05.08.15.00
Serial Number: 725112060578
Position camera and hit enter key to take image
0 / 6

The app runs with a simple UI. The main window displays the image from RGB imager, the lower left corner displays the left and right images. In case the device does not have RGB, only the left and right images display on the lower left corner.

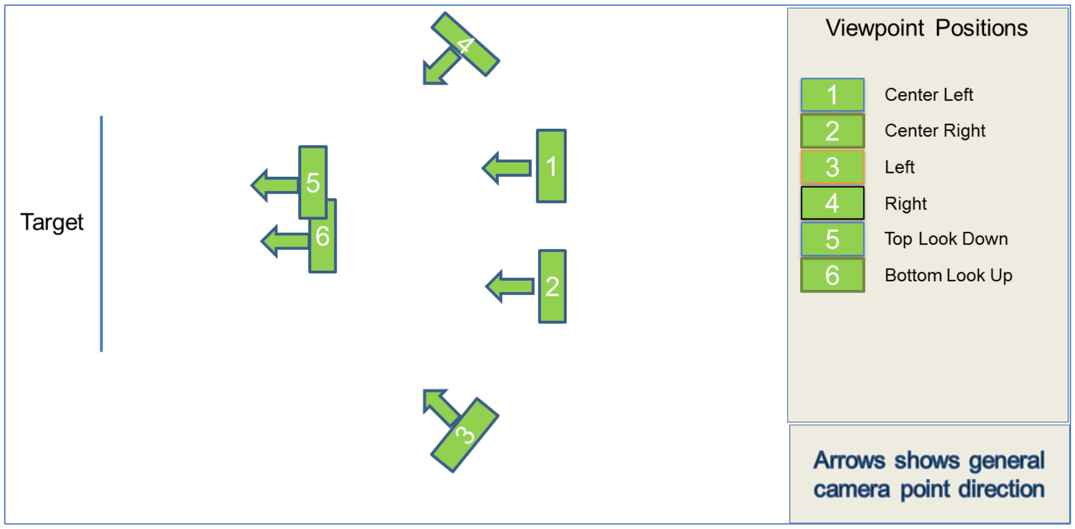
A few text messages are overlay on top of the images:

- On the top left corner, it display the device model name, for example, Intel RealSense 415.
- On the top right corner, it displays the FW version and serial number of the device.
- On the bottom right corner, it displays a progress counter in the form of x/6 where x is the number of images captured and accepted out of the total 6 images required.
- In the middle of the Window is a green help message to instruct the user to position the camera and press the enter key to capture image.
- In the top middle portion of the Window is an area where error message will display in red when the image is not accepted or other error conditions.

### 4.3.2 Capturing Images from 6 Viewpoints

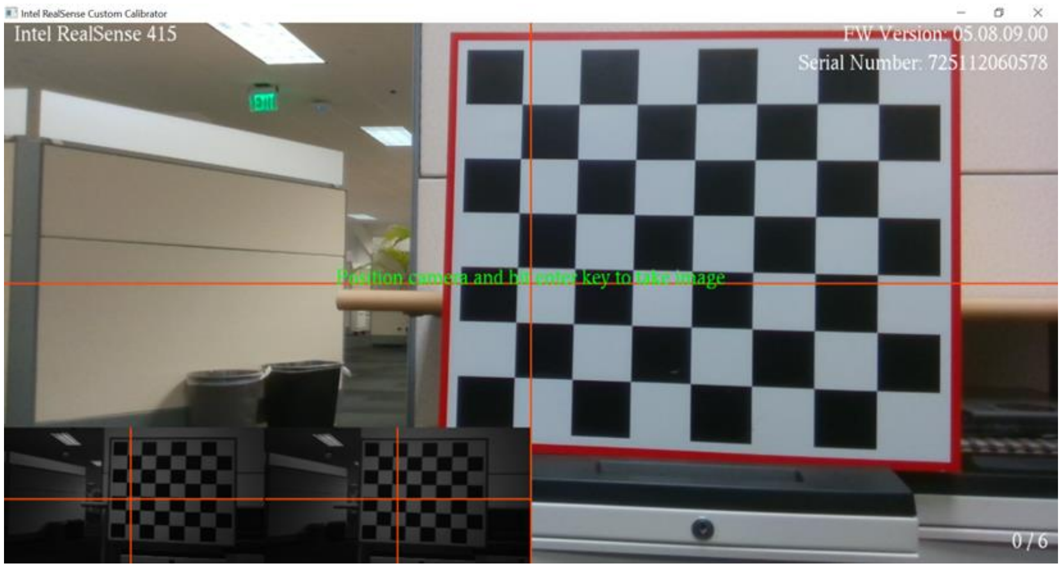
The calibration algorithm in the Custom Calibration Sample Application requires 6 images of the target from different viewpoints. This is the minimum number of images for simplicity. The application UI will guide the user to go through the image capturing process.

- **Viewpoints** - the choice of the viewpoint is critical to the accuracy of the calibration results. A general rule is that the target in these 6 images combined should cover as much as possible of the entire field of view and from different angles and distances from the target.

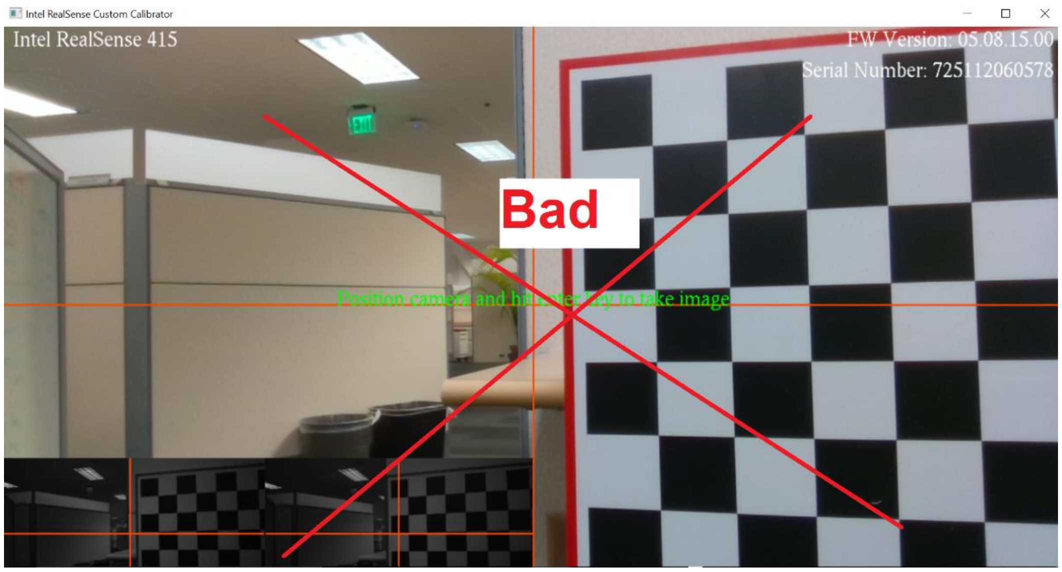


- **Cover whole target** - images from any of the viewpoints should cover the entire target in all imagers (Left, Right and RGB).

Good example – cover the entire target in all three imagers:


Intel RealSense Custom Calibrator
Intel RealSense 415
FW Version: 05.08.09.00
Serial Number: 725112060578
Position camera and hit enter key to take image
0 / 6

Bad example - cover only portion of the target in any of the three imagers:


Intel RealSense Custom Calibrator
Intel RealSense 415
FW Version: 05.08.15.00
Serial Number: 725112060578
Bad
Position camera and hit enter key to take image

Now, let’s capture the images.

### 4.3.2.1 Viewpoint #1 – Center Right

Adjust the tripod and position the device on the center right facing directly to the target.

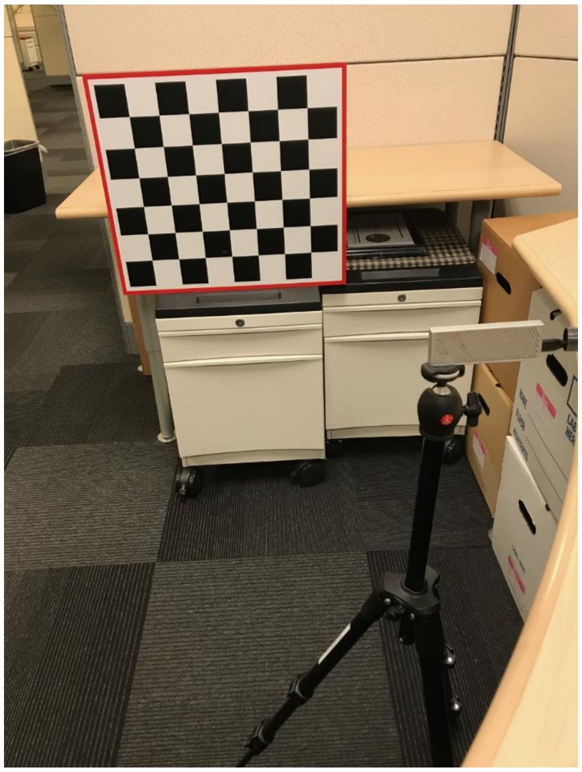
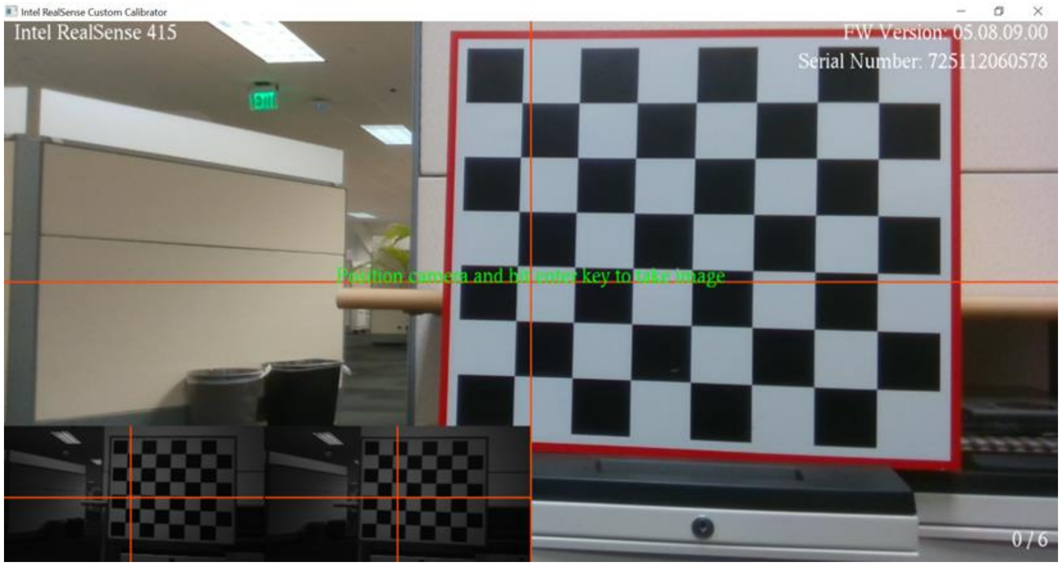


Figure 4-1 Center Right Position

The viewpoint should looks like below with the target cover much of the right side the FOV.



Intel RealSense Custom Calibrator

Intel RealSense 415

FW Version: 05.08.09.00

Serial Number: 725112060578

Position camera and hit enter key to take image

0 / 6

**Figure 4-2 Viewpoint #1**

Press enter to capture the image. If successful, the frame counter on the lower right corner will increase to 1/6 meaning 1 out of 6 images are captured. If it failed, a red error message will appear and user is directed to retake a viewpoint.

### 4.3.2.2 Viewpoint #2 - Center Left

Move the device to the center left directly facing the target so the target covers much of the left portion of the FOV.

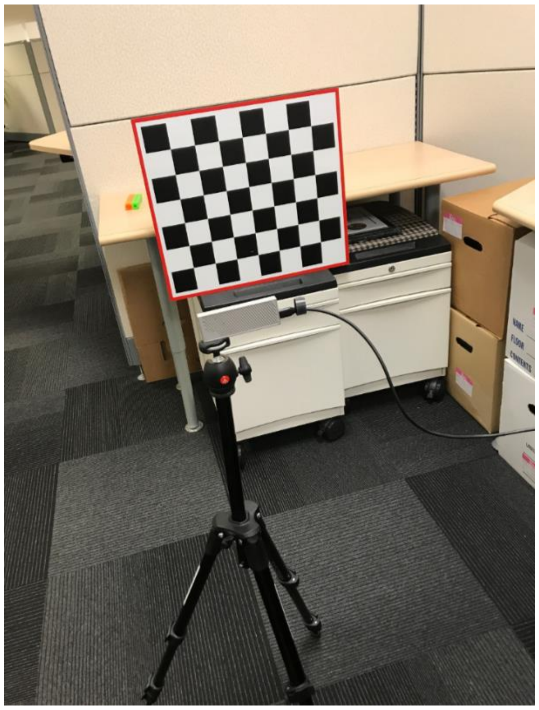


**Figure 4-3 Center Right**

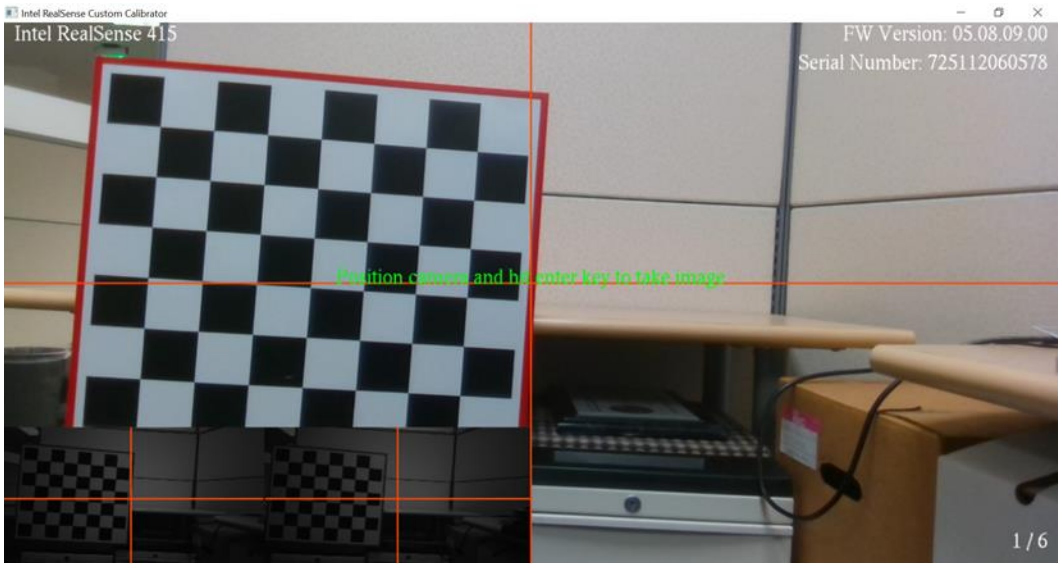

Intel RealSense Custom Calibrator
Intel RealSense 415
FW Version: 05.08.09.00
Serial Number: 725112060578
Position camera and hit enter key to take image
1 / 6

**Figure 4-4 Viewpoint #2**

Press enter to capture the image. If successful, the frame counter on the lower right corner will increase to 2/6 meaning 2 out of 6 images are captured. If it failed, a red error message will appear and user is directed to retake a viewpoint.

#### 4.3.2.3 Viewpoint #3 - Left

Move the device to the left of the camera facing the device at an angle. Use the viewpoint below for reference.

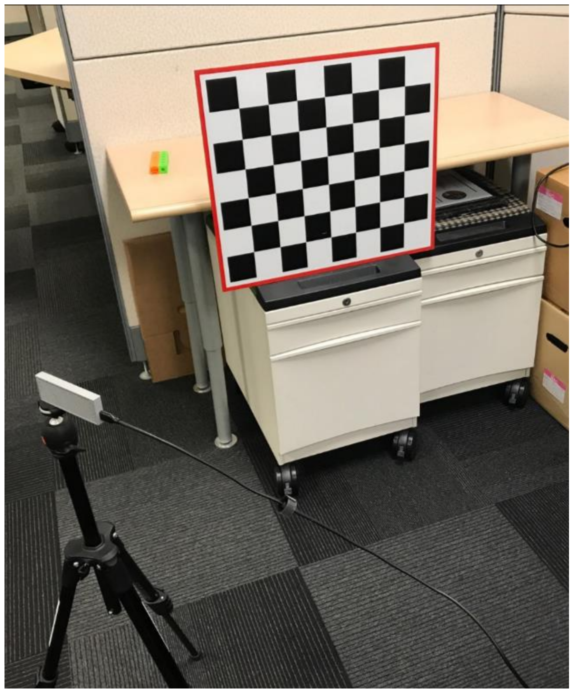
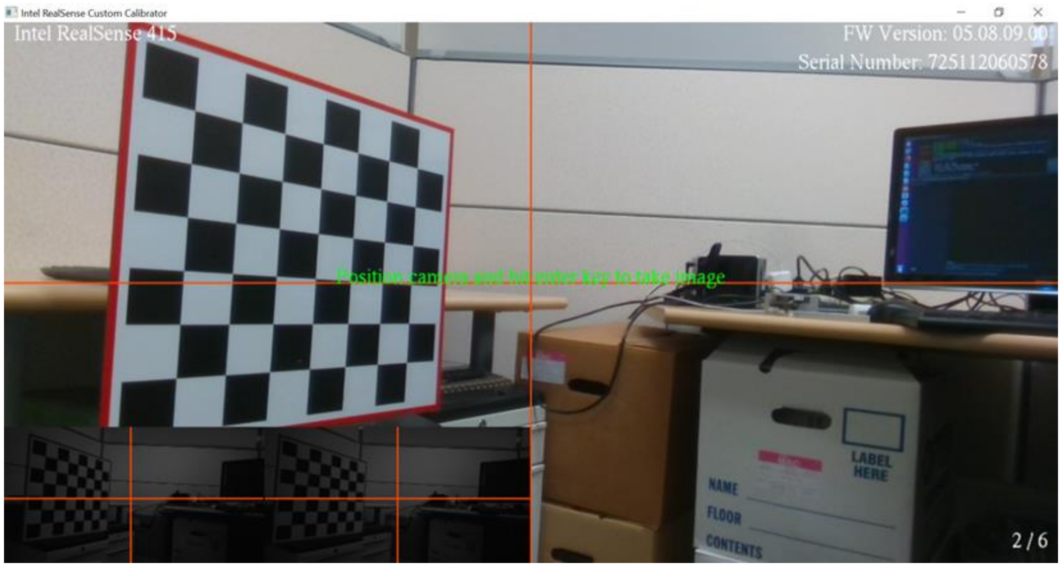


**Figure 4-5 Left**



Intel RealSense 415

FW Version: 05.08.09.00

Serial Number: 725112060578

Position camera and hit enter key to take image

2 / 6

**Figure 4-6 Viewpoint #3**

Press enter to capture the image. If successful, the frame counter on the lower right corner will increase to 3/6 meaning 3 out of 6 images are captured. If it failed, a red error message will appear and user is directed to retake a viewpoint.

### 4.3.2.4 Viewpoint #4 – Right

Move the device to the right facing the target at an angle. Use the viewpoint below for reference.

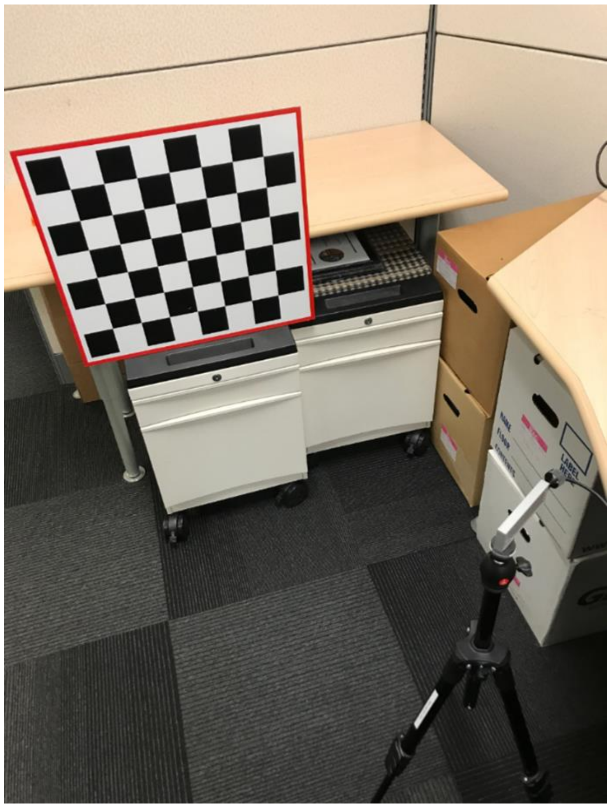


Figure 4-7 Right

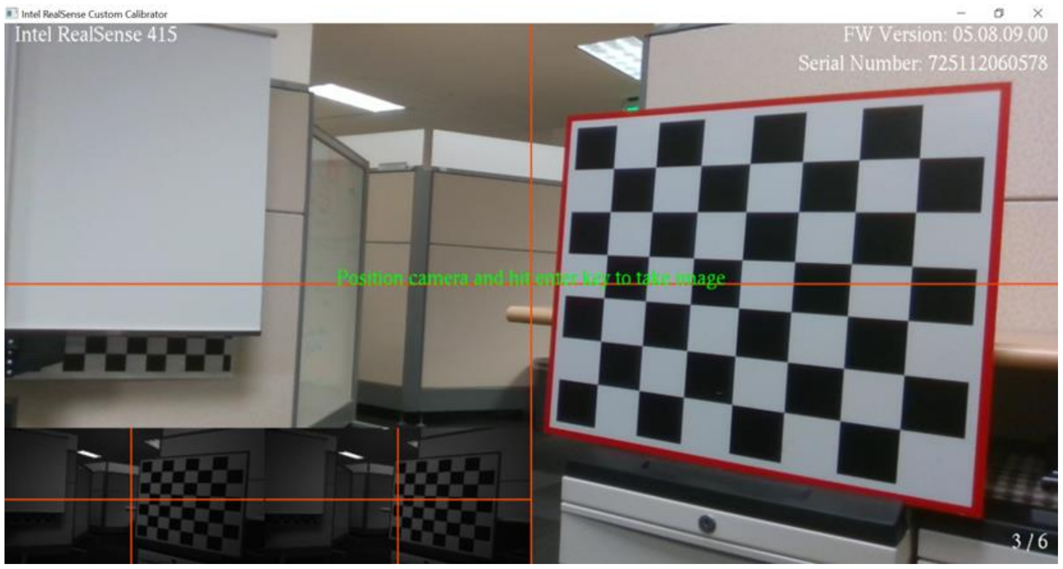

Intel RealSense Custom Calibrator
Intel RealSense 415
FW Version: 05.08.09.00
Serial Number: 725112060578
Position camera and hit enter key to take image
3 / 6

**Figure 4-8 Viewpoint #4**

Press enter to capture the image. If successful, the frame counter on the lower right corner will increase to 4/6 meaning 4 out of 6 images are captured. If it failed, a red error message will appear and user is directed to retake a viewpoint.

### 4.3.2.5 Viewpoint #5 – Top Looking Down

Adjust the tripod so the device high enough to look down the target at an angle. Use the viewpoint below for reference.

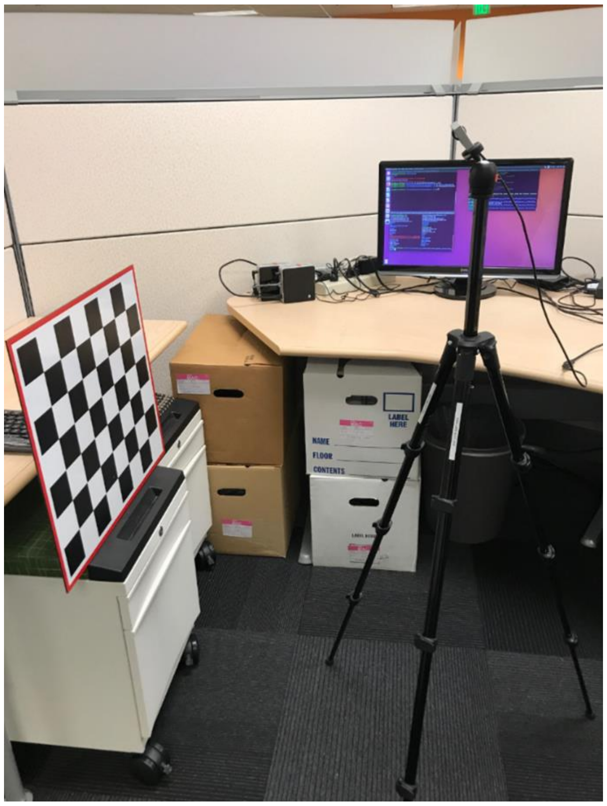


**Figure 4-9 Top Looking Down**

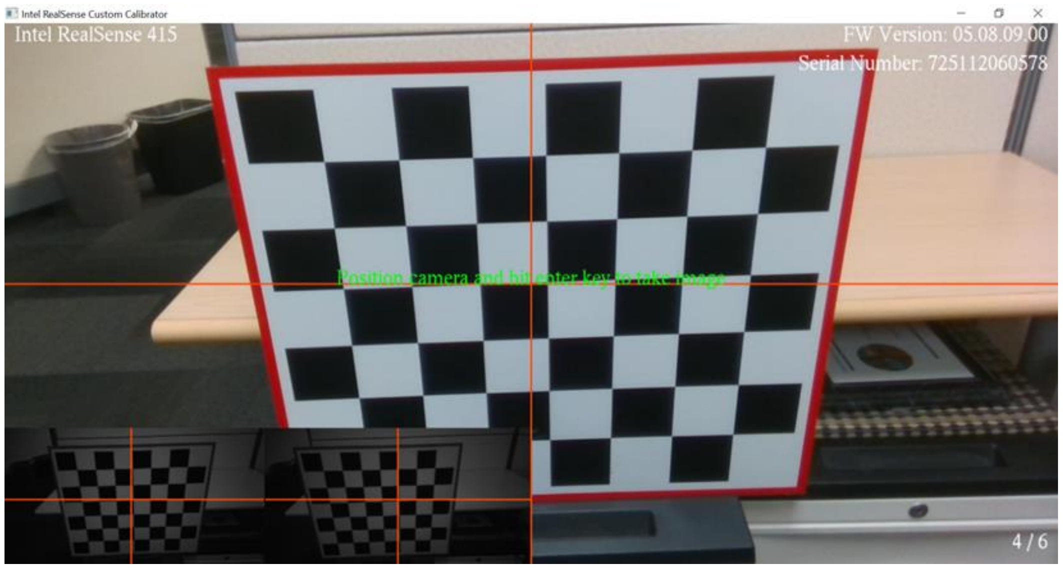

Intel RealSense Custom Calibrator
Intel RealSense 415
FW Version: 05.08.09.00
Serial Number: 725112060578
Position camera and hit enter key to take image
4 / 6

**Figure 4-10 Viewpoint #5**

Press enter to capture the image. If successful, the frame counter on the lower right corner will increase to 5/6 meaning 5 out of 6 images are captured. If it failed, a red error message will appear and user is directed to retake a viewpoint.

### 4.3.2.6 Viewpoint #6 – Bottom Looking Up

Folding the tripod so that the device will be in a very low position looking up to the target. Use the viewpoint below for reference.

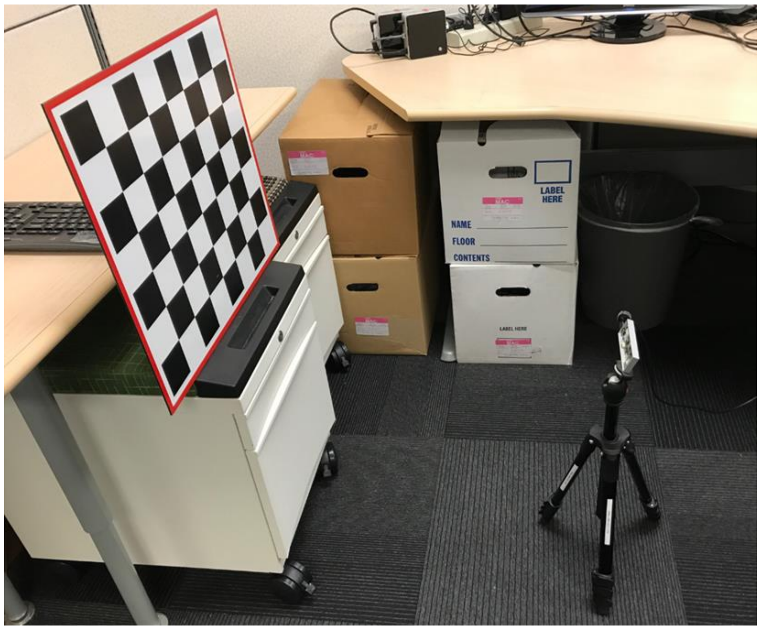


**Figure 4-11 Bottom Looking Up**

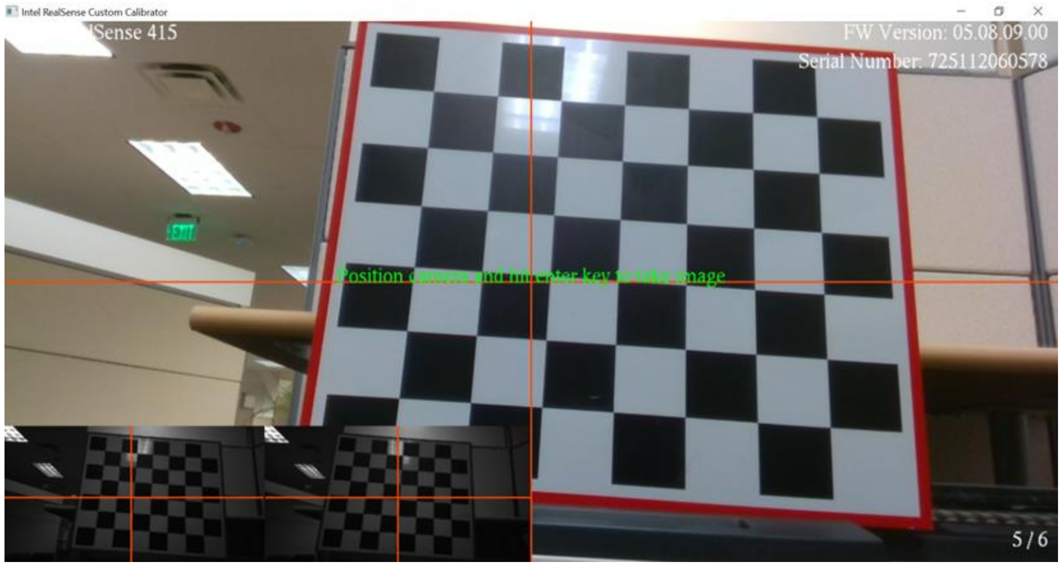

Intel RealSense Custom Calibrator
Sense 415
FW Version: 05.08.09.00
Serial Number: 725112060578
Position camera and hit enter key to take image
5 / 6

**Figure 4-12 Viewpoint #6**

Press enter to capture the image. If successful, the frame counter on the lower right corner will increase to 6/6 meaning 6 out of 6 images are captured. If it failed, a red error message will appear and user is directed to retake a viewpoint.

## 4.4 Calibration Result

Once all 6 images captured, calibration will run and should finish very quickly. The results usually located under the CustomCalibration\CustomResult folder with device serial number as the folder name. For example, CustomCalibration\CustomResult\725112060578\DC1

**6 RGB images**:

colorImage001.png
colorImage002.png
colorImage003.png
colorImage004.png
colorImage005.png
colorImage006.png

**6 left images:**

leftImage001.png
leftImage002.png
leftImage003.png
leftImage004.png
leftImage005.png
leftImage006.png

**Right RGB Images**

rightImage001.png
rightImage002.png
rightImage003.png
rightImage004.png
rightImage005.png
rightImage006.png

**Optimized calibration parameters in XML:**

CalibrationParameters.xml

```
<?xml version="1.0"?>
<Config>
   <param name = "ResolutionLeftRight">
      <value>1920</value>
      <value>1080</value>
   </param>
   <param name = "FocalLengthLeft">
      <value>1378.69</value>
      <value>1379.13</value>
   </param>
```

```
<param name = "PrincipalPointLeft">
    <value>965.562</value>
    <value>542.603</value>
</param>
<param name = "DistortionLeft">
    <value>0.09689</value>
    <value>-0.0235634</value>
    <value>-5.05513e-05</value>
    <value>0.000105855</value>
    <value>-0.640377</value>
</param>
<param name = "FocalLengthRight">
    <value>1389.55</value>
    <value>1390.1</value>
</param>
<param name = "PrincipalPointRight">
    <value>957.402</value>
    <value>553.108</value>
</param>
<param name = "DistortionRight">
    <value>0.123289</value>
    <value>-0.290911</value>
    <value>0.000149083</value>
    <value>0.000723527</value>
    <value>0.0992178</value>
</param>
<param name = "RotationLeftRight">
    <value>0.999983</value>
    <value>0.00150008</value>
    <value>-0.00558928</value>
    <value>-0.00150249</value>
    <value>0.999999</value>
    <value>-0.000428313</value>
    <value>0.00558863</value>
    <value>0.000436704</value>
    <value>0.999984</value>
</param>
<param name = "TranslationLeftRight">
    <value>-55.3888</value>
    <value>0.0448052</value>
    <value>1.52189</value>
</param>
<param name = "HasRGB">
    <value>1</value>
</param>
<param name = "ResolutionRGB">
    <value>1920</value>
    <value>1080</value>
</param>
<param name = "FocalLengthRGB">
    <value>1376.09</value>
    <value>1376.4</value>
    </param>
    <param name = "PrincipalPointRGB">
        <value>947.958</value>
        <value>531.068</value>
    </param>
    <param name = "DistortionRGB">
        <value>0.109862</value>
        <value>-0.122685</value>
        <value>-0.000453445</value>
        <value>-8.61456e-05</value>
        <value>-0.396171</value>
    </param>
    <param name = "RotationLeftRGB">
        <value>0.999994</value>
        <value>0.00346622</value>
        <value>0.000876088</value>
        <value>-0.00346642</value>
        <value>0.999994</value>
        <value>0.000222113</value>
        <value>-0.000875313</value>
        <value>-0.000225148</value>
        <value>1</value>
    </param>
    <param name = "TranslationLeftRGB">
        <value>14.6987</value>
        <value>-0.0644999</value>
        <value>0.448085</value>
    </param>
</Config>
```

## 4.5 Updating Results to Device

We will use CustomRW to read the CalibrationParameters.xml and write the parameters to the device. Checking the depth quality before and after updating the calibration data on the device.

### 4.5.1 Depth Quality Check before Updating Calibration

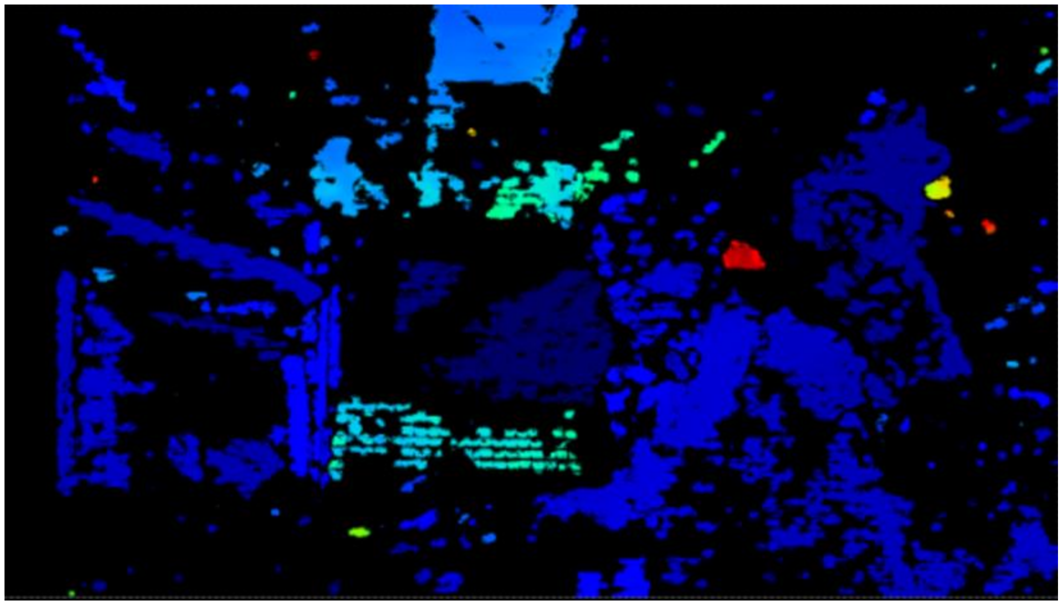


**Figure 4-13 Depth Quality before Updating Calibration**

### 4.5.2 Writing Optimized Calibration to Device

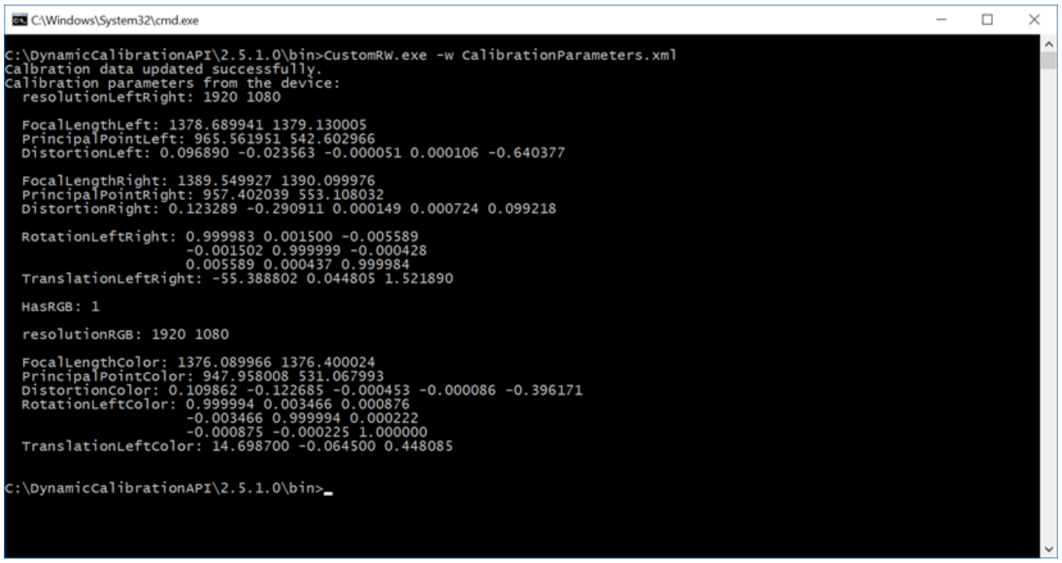
Cd C:\DynamicCalibrationAPI\2.5.1.0\bin

Dump the current parameters

CustomRW –r > before.txt

Writing the parameters in the XML result file into device

CustomRW –w CalibrationParameters.xml



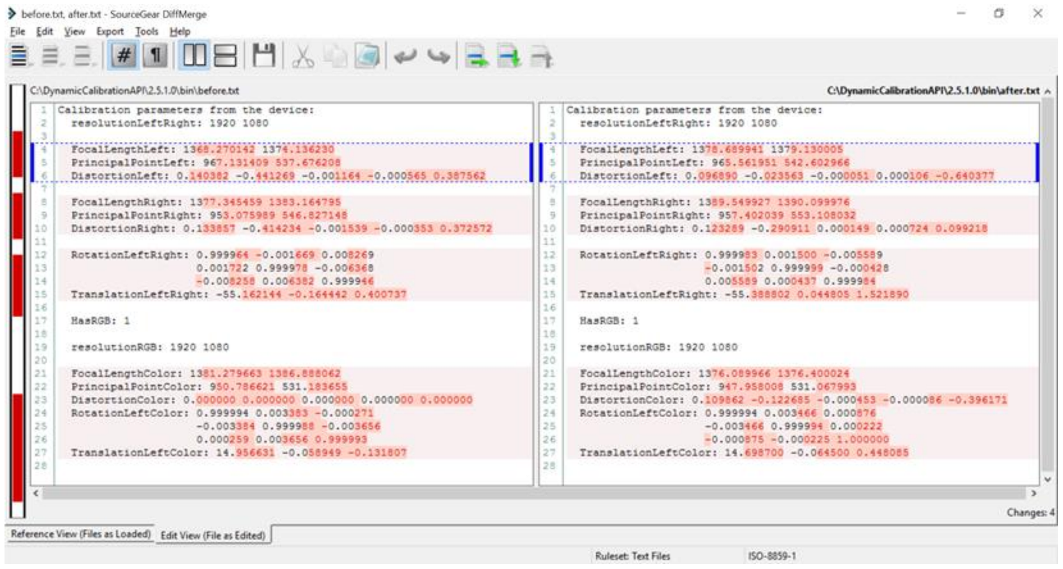
```
C:\DynamicCalibrationAPI\2.5.1.0\bin>CustomRW.exe -w CalibrationParameters.xml
Calbration data updated successfully.
Calibration parameters from the device:
  resolutionLeftRight: 1920 1080

  FocalLengthLeft: 1378.689941 1379.130005
  PrincipalPointLeft: 965.561951 542.602966
  DistortionLeft: 0.096890 -0.023563 -0.000051 0.000106 -0.640377

  FocalLengthRight: 1389.549927 1390.099976
  PrincipalPointRight: 957.402039 553.108032
  DistortionRight: 0.123289 -0.290911 0.000149 0.000724 0.099218

  RotationLeftRight: 0.999983 0.001500 -0.005589
                     -0.001502 0.999999 -0.000428
                     0.005589 0.000437 0.999984
  TranslationLeftRight: -55.388802 0.044805 1.521890

  HasRGB: 1

  resolutionRGB: 1920 1080

  FocalLengthColor: 1376.089966 1376.400024
  PrincipalPointColor: 947.958008 531.067993
  DistortionColor: 0.109862 -0.122685 -0.000453 -0.000086 -0.396171
  RotationLeftColor: 0.999994 0.003466 0.000876
                     -0.003466 0.999994 0.000222
                     -0.000875 -0.000225 1.000000
  TranslationLeftColor: 14.698700 -0.064500 0.448085


C:\DynamicCalibrationAPI\2.5.1.0\bin>_
```

**Figure 4-14 Updating Calibration Parameters to Device**

Dump the parameters from device:

CustomRW –r > after.txt

Compare the parameters changed:


before.txt, after.txt - SourceGear DiffMerge
File Edit View Export Tools Help
Reference View (Files as Loaded)
Edit View (File as Edited)
Changes: 4
Ruleset: Text Files
ISO-8859-1

C:\DynamicCalibrationAPI\2.5.1.0\bin\before.txt

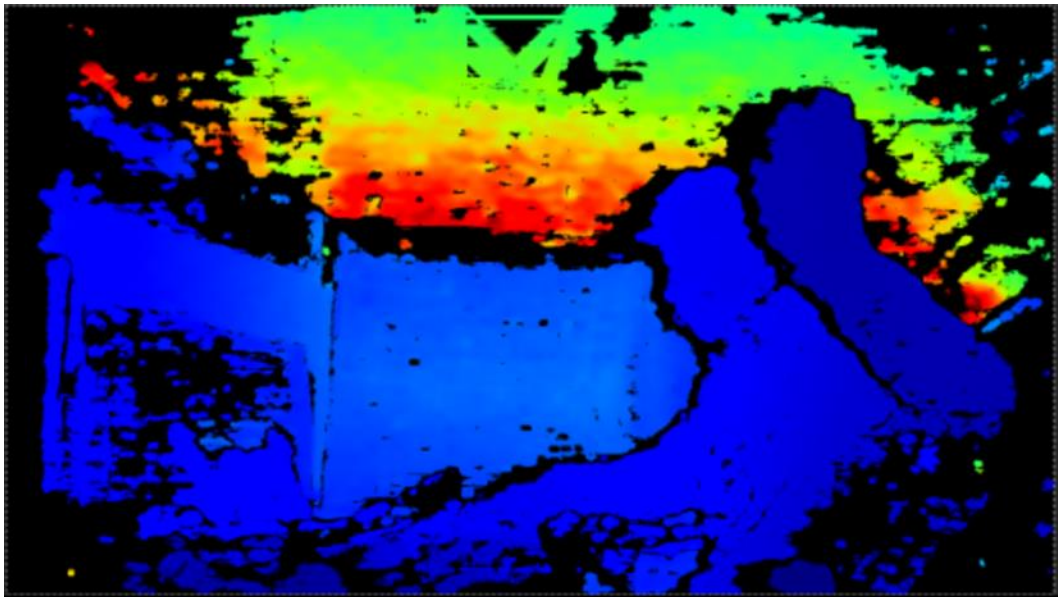
```
Calibration parameters from the device:
  resolutionLeftRight: 1920 1080

  FocalLengthLeft: 1368.270142 1374.136230
  PrincipalPointLeft: 967.131409 537.676208
  DistortionLeft: 0.140382 -0.441269 -0.001164 -0.000565 0.387562

  FocalLengthRight: 1377.345459 1383.164795
  PrincipalPointRight: 953.075989 546.827148
  DistortionRight: 0.133857 -0.414234 -0.001539 -0.000353 0.372572

  RotationLeftRight: 0.999964 -0.001669 0.008269
                     0.001722 0.999978 -0.006368
                     -0.008258 0.006382 0.999946
  TranslationLeftRight: -55.162144 -0.164442 0.400737

  HasRGB: 1

  resolutionRGB: 1920 1080

  FocalLengthColor: 1381.279663 1386.888062
  PrincipalPointColor: 950.786621 531.183655
  DistortionColor: 0.000000 0.000000 0.000000 0.000000 0.000000
  RotationLeftColor: 0.999994 0.003383 -0.000271
                     -0.003384 0.999988 -0.003656
                     0.000259 0.003656 0.999993
  TranslationLeftColor: 14.956631 -0.058949 -0.131807

```

C:\DynamicCalibrationAPI\2.5.1.0\bin\after.txt

```
Calibration parameters from the device:
  resolutionLeftRight: 1920 1080

  FocalLengthLeft: 1378.689941 1379.130005
  PrincipalPointLeft: 965.561951 542.602966
  DistortionLeft: 0.096890 -0.023563 -0.000051 0.000106 -0.640377

  FocalLengthRight: 1389.549927 1390.099976
  PrincipalPointRight: 957.402039 553.108032
  DistortionRight: 0.123289 -0.290911 0.000149 0.000724 0.099218

  RotationLeftRight: 0.999983 0.001500 -0.005589
                     -0.001502 0.999999 -0.000428
                     0.005589 0.000437 0.999984
  TranslationLeftRight: -55.388802 0.044805 1.521890

  HasRGB: 1

  resolutionRGB: 1920 1080

  FocalLengthColor: 1376.089966 1376.400024
  PrincipalPointColor: 947.958008 531.067993
  DistortionColor: 0.109862 -0.122685 -0.000453 -0.000086 -0.396171
  RotationLeftColor: 0.999994 0.003466 0.000876
                     -0.003466 0.999994 0.000222
                     -0.000875 -0.000225 1.000000
  TranslationLeftColor: 14.698700 -0.064500 0.448085

```

**Figure 4-15 Calibration Parameter Change**

## 4.5.3 Depth Quality Check after Updating Calibration



# 5 Developing Custom Calibration Application

This section demonstrates how to develop custom calibration application through key aspects of the sample app we used in *Calibrating Device with Custom Calibration Sample Application.* The sample app uses OpenCV algo as example.

The general process is to configure the Intel® RealSense™ device for calibration, capture images from the camera module, perform the desired calibration computations with OpenCV, and to write the new calibration parameters back to the cameras using routines from the Intel® RealSense™ calibration library.

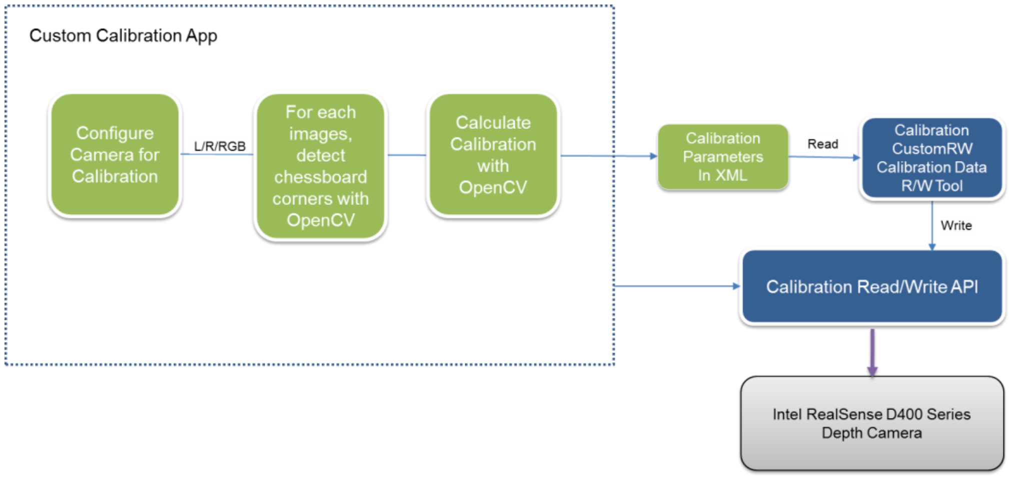

Custom Calibration App
Configure Camera for Calibration
L/R/RGB
For each images, detect chessboard corners with OpenCV
Calculate Calibration with OpenCV
Calibration Parameters In XML
Read
Calibration CustomRW Calibration Data R/W Tool
Write
Calibration Read/Write API
Intel RealSense D400 Series Depth Camera

**Figure 5-1 General Process for custom Calibration**

## 5.1 Calibration Mode Camera Configuration

For calibration, the camera device needs to be configured to capture calibration images for both depth and RGB camera using LibRealSense.

### 5.1.1 Emitter

The emitter should be turned off during calibration to avoid interference.

Sample code in Rs400Device:

```
        m_rsDevice->EnableEmitter(0.0f);
```

### 5.1.2 Auto Exposure

Auto exposure should be turn on and the AE setpoint be adjusted according to lighting condition. In indoor room lighting, a setpoint value around 500 – 800 should be sufficient. In outdoor, 1200 or higher.

```
        m_rsDevice->EnableAutoExposure(1.0f);
        if (m_cameraInfo.isWide)
                m_rsDevice->SetAeControl(800);
        else
                m_rsDevice->SetAeControl(500);
```

### 5.1.3 Streaming Resolution and Format

Calibration frame resolutions and formats are described in *Table 2-1 Frame Formats Used in Custom Calibration, for example, in this sample, or D415 device, the resolution is 1920x1080 with frame rate of 15 fps. The L/R format is Y16 and the RGB format is YUY2.*

```
        m_rsDevice->SetMediaMode(m_width, m_height, m_fps, m_rgbWidth,
m_rgbHeight, m_cameraInfo.isRGB);

        m_rsDevice->StartCapture([&](const void *leftImage, const void *rightImage,
                const void *colorImage, const uint64_t timeStamp)
```

### 5.1.4 Demosaic Left/Right Images for ASR / PSR SKUs

For ASR/PSR SKUs, left and right images need to be demosaiced.

## 5.2 Detecting the Chessboard in an Image with OpenCV

For each of the captured images, find the chessboard corners with OpenCV findChessboardCorners.

```
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/calib3d.hpp>
#include <vector>

using namespace std;
using namespace cv;

bool DetectChessboard(const Mat& image, const Size& chessboardSize, vector<Point2f>&
corners)
{
        // Find chessboard corners
        if (!findChessboardCorners(image, chessboardSize, corners,
CALIB_CB_ADAPTIVE_THRESH | CALIB_CB_NORMALIZE_IMAGE | CALIB_CB_FILTER_QUADS))
                return false;

        // Refine them
        cornerSubPix(image, corners, Size(11, 11), Size(-1, -1),
TermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
        return true;
}
```

## 5.3 Calculating Depth Camera Calibration with OpenCV

```
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/calib3d.hpp>
#include <vector>

using namespace std;
using namespace cv;

static void CreateCorners3D(const Size& chessboardSize, float checkerSize, size_t
numImages, vector<vector<Point3f> >& corners3D)
{
        corners3D.resize(numImages);
        corners3D[0].resize(chessboardSize.width * chessboardSize.height);

        for (int i = 0; i < chessboardSize.height; i++)
                for (int j = 0; j < chessboardSize.width; j++)
                        corners3D[0][i * chessboardSize.width + j] = Point3f(j *
checkerSize, i * checkerSize, 0.0f);

        for (size_t i = 1; i < numImages; i++)
                        corners3D[i] = corners3D[0];
}

double CalibrateDepthCamera(const vector<vector<Point2f> >& cornersLeft, const
vector<vector<Point2f> >& cornersRight, const Size& chessboardSize, float checkerSize,
const Size& imageSizeLR, Mat& Kl, Mat& Dl, Mat& Kr, Mat& Dr, Mat& Rlr, Mat& Tlr)
{
        CV_Assert(cornersLeft.size() != 0 && cornersLeft.size() == cornersRight.size());
        CV_Assert(checkerSize > 0.0f);

        // Create 3D prototype of the corners
        vector<vector<Point3f> > corners3D;
        CreateCorners3D(chessboardSize, checkerSize, cornersLeft.size(), corners3D);

        // Calibrate each camera individualy
        calibrateCamera(corners3D, cornersLeft, imageSizeLR, Kl, Dl, noArray(),
noArray(), CV_CALIB_FIX_ASPECT_RATIO, TermCriteria(TermCriteria::COUNT +
TermCriteria::EPS, 60, DBL_EPSILON));
        calibrateCamera(corners3D, cornersRight, imageSizeLR, Kr, Dr, noArray(),
noArray(), CV_CALIB_FIX_ASPECT_RATIO, TermCriteria(TermCriteria::COUNT +
TermCriteria::EPS, 60, DBL_EPSILON));

        // Calibrate the extrinsics between them
        return stereoCalibrate(corners3D, cornersLeft, cornersRight, Kl, Dl, Kr, Dr, 
Size(-1, -1), Rlr, Tlr, noArray(), noArray(), CV_CALIB_FIX_INTRINSIC | 
CV_CALIB_USE_INTRINSIC_GUESS, TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 300, 
DBL_EPSILON));
}
```

## 5.4 Calculating RGB Camera Calibration with OpenCV

***Note: Assumes good depth camera calibration)***

```
double CalibrateRGBCamera(const vector<vector<Point2f> >& cornersLeft, const 
vector<vector<Point2f> >& cornersRGB, const Size& chessboardSize, float checkerSize, 
const Size& imageSizeRGB, const Mat& Kl, const Mat& Dl, Mat& Kc, Mat& Dc, Mat& Rlc, Mat& 
Tlc)
{
        CV_Assert(cornersLeft.size() != 0 && cornersLeft.size() == cornersRGB.size());
        CV_Assert(checkerSize > 0.0f);

        // Create 3D prototype of the corners
        vector<vector<Point3f> > corners3D;
        CreateCorners3D(chessboardSize, checkerSize, cornersLeft.size(), corners3D);

        // Calibrate RGB camera
        calibrateCamera(corners3D, cornersRGB, imageSizeRGB, Kc, Dc, noArray(), 
noArray(), CV_CALIB_FIX_ASPECT_RATIO, TermCriteria(TermCriteria::COUNT + 
TermCriteria::EPS, 60, DBL_EPSILON));

        // Calibrate the extrinsics between them
        return stereoCalibrate(corners3D, cornersLeft, cornersRGB, Kl, Dl, Kc, Dc, Size(-
1, -1), Rlc, Tlc, noArray(), noArray(), CV_CALIB_FIX_INTRINSIC | 
CV_CALIB_USE_INTRINSIC_GUESS, TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 300, 
DBL_EPSILON));
}
```

## 5.5 Calculating RGB Camera Calibration Extrinsics with OpenCV

***Note: Assumes good depth camera calibration and intrinsics for the RGB camera.***

The **`CustomCalibration.h`** header file in the sample code contained the following unimplemented prototype:

```
double RecalibrateRGBCamera(const std::vector<std::vector<cv::Point2f> >& cornersLeft, 
const std::vector<std::vector<cv::Point2f> >& cornersRGB, const cv::Size& chessboardSize, 
float checkerSize, const cv::Mat& Kl, const cv::Mat& Dl, const cv::Mat& Kc, const 
cv::Mat& Dc, cv::Mat& Rlc, cv::Mat& Tlc);
```

## 5.6 Writing Calibration Parameters

A user custom calibration app can choose one of the two approaches to update the results to the device:

- To link to the WriteCustomCalibrationParameters and ReadCalibrationParameters and write directly to the device through the APIs.
- To write the results into a parameter XML file and then use CustomRW to read/write the parameters to the device.

# Appendix

# 6 Sample Application Source Code

## 6.1 Project Files & Makefiles

### 6.1.1 CustomCalibrationTest.sln

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 15
VisualStudioVersion = 15.0.26430.16
MinimumVisualStudioVersion = 10.0.40219.1
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "CustomCalibrationTest",
"CustomCalibrationTest.vcxproj", "{56DE6038-E9C7-41D8-A705-2DA5B20C778D}"
EndProject
Global
	GlobalSection(SolutionConfigurationPlatforms) = preSolution
		Debug|x64 = Debug|x64
		Debug|x86 = Debug|x86
		Release|x64 = Release|x64
		Release|x86 = Release|x86
	EndGlobalSection
	GlobalSection(ProjectConfigurationPlatforms) = postSolution
		{56DE6038-E9C7-41D8-A705-2DA5B20C778D}.Debug|x64.ActiveCfg = Debug|x64
		{56DE6038-E9C7-41D8-A705-2DA5B20C778D}.Debug|x64.Build.0 = Debug|x64
		{56DE6038-E9C7-41D8-A705-2DA5B20C778D}.Debug|x86.ActiveCfg = Debug|Win32
		{56DE6038-E9C7-41D8-A705-2DA5B20C778D}.Debug|x86.Build.0 = Debug|Win32
		{56DE6038-E9C7-41D8-A705-2DA5B20C778D}.Release|x64.ActiveCfg =
Release|x64
		{56DE6038-E9C7-41D8-A705-2DA5B20C778D}.Release|x64.Build.0 = Release|x64
		{56DE6038-E9C7-41D8-A705-2DA5B20C778D}.Release|x86.ActiveCfg =
Release|Win32
		{56DE6038-E9C7-41D8-A705-2DA5B20C778D}.Release|x86.Build.0 =
Release|Win32
	EndGlobalSection
	GlobalSection(SolutionProperties) = preSolution
		HideSolutionNode = FALSE
	EndGlobalSection
EndGlobal
```

### 6.1.2 Makefile

```
ifneq ($(wildcard ../../make.include),)
  include ../../make.include
else
  include ../make.include
endif

TRG = CustomCalibrationTest$(SUFFIX)

ifneq ($(wildcard ../bin),)
```

```
  OUT = ../bin/$(TRG)
else
  OUT = $(TRG)
endif

COBJDIR = obj-c
CPPOBJDIR = obj-cpp

CXXFLAGS += -m64 -pipe -std=gnu++11 -W

IDIR =  -I ./Include    \
        -I ../Include    \
        -I ../Include/DSDynamicCalibrationAPI \
        -I ../librealsense/include  \
        -I ../OpenCV\OpenCV330\include


CXXFLAGS += $(IDIR)

CPPSRC = \
        CalibrationManager.cpp  \
        CustomCalibration.cpp   \
        CustomCalibrationWrapper.cpp  \
        Rs400Device.cpp   \
        Main.cpp \

CPPOBJS = $(addprefix $(CPPOBJDIR)/, $(notdir $(CPPSRC:.cpp=.o)))

LDIR = -L ../librealsense/lib -L ../Lib/DSDynamicCalibrationAPI -L
../OpenCV/OpenCV330/lib
ifneq ($(wildcard ../lib),)
  LDIR += -L ../lib
endif

LIBS = -lDSDynamicCalibrationAPI -lrealsense2 -lglut -lGL -lpthread -lrt
LIBS += -l:libopencv_core.so.3.3.0 -l:libopencv_imgproc.so.3.3.0 -
l:libopencv_features2d.so.3.3.0 -l:libopencv_calib3d.so.3.3.0

all: prepare $(OUT)
        @echo "\n\tBuild completed\n"


$(CPPOBJDIR)/%.o: %.cpp
        $(CXX) $(CXXFLAGS) -c -o $@ $<

$(OUT): $(CPPOBJS)
        $(CXX) $(CXXFLAGS) $^ $(LDIR) $(LIBS) -o $@

prepare:
        mkdir -p $(CPPOBJDIR)

install:
         cp ../Lib/DSDynamicCalibrationAPI/libDSDynamicCalibrationAPI.so /usr/local/lib
         cp ../OpenCV/OpenCV330/lib/*.so.3.3.0 /usr/local/lib
         cp ../librealsense/lib/*.so.2.8.0 /usr/local/lib
         ldconfig

uninstall:
        rm -f /usr/local/lib/libDSDynamicCalibrationAPI.so
        rm -f /usr/local/lib/libopencv*.so.3.3.0
        rm -f /usr/local/lib/librealsense2.so.2.8.0
        ldconfig

clean:
        rm -f $(CPPOBJS) $(OUT)
        rm -fr $(CPPOBJDIR)

.PHONY: clean prepare
```

## 6.2 Include Files

### 6.2.1 CustomCalibration.h

```
/*
 * INTEL CORPORATION PROPRIETARY INFORMATION
 * This software is supplied under the terms of a license agreement
 * or nondisclosure agreement with Intel Corporation and may not be
 * copied or disclosed except in accordance with the terms of that
 * agreement.
 * Copyright(c) 2016-2017 Intel Corporation. All Rights Reserved.
 */

#ifndef CUSTOMCALIBRATION_H_
#define CUSTOMCALIBRATION_H_

#include <opencv2/opencv.hpp>

// Sample code that detects the chessboard from an image using OpenCV
bool DetectChessboard(const cv::Mat& image, const cv::Size& chessboardSize,
std::vector<cv::Point2f>& corners);

// Sample code how to calculate depth camera calibration using OpenCV
double CalibrateDepthCamera(const std::vector<std::vector<cv::Point2f> >& cornersLeft,
const std::vector<std::vector<cv::Point2f> >& cornersRight, const cv::Size&
chessboardSize, float checkerSize, const cv::Size& imageSizeLR, cv::Mat& Kl, cv::Mat& Dl,
cv::Mat& Kr, cv::Mat& Dr, cv::Mat& Rlr, cv::Mat& Tlr);

// Sample code how to calculate RGB camera calibration using OpenCV (this assumes good
depth camera calibration)
double CalibrateRGBCamera(const std::vector<std::vector<cv::Point2f> >& cornersLeft,
const std::vector<std::vector<cv::Point2f> >& cornersRGB, const cv::Size& chessboardSize,
float checkerSize, const cv::Size& imageSizeRGB, const cv::Mat& Kl, const cv::Mat& Dl,
cv::Mat& Kc, cv::Mat& Dc, cv::Mat& Rlc, cv::Mat& Tlc);

// Sample code how to recalculate RGB camera calibration extrinsics using OpenCV (this
assumes good depth camera calibration and instrinsics of the RGB camera)
double RecalibrateRGBCamera(const std::vector<std::vector<cv::Point2f> >& cornersLeft,
const std::vector<std::vector<cv::Point2f> >& cornersRGB, const cv::Size& chessboardSize,
float checkerSize, const cv::Mat& Kl, const cv::Mat& Dl, const cv::Mat& Kc, const
cv::Mat& Dc, cv::Mat& Rlc, cv::Mat& Tlc);

#endif //CUSTOMCALIBRATION_H_
```

### 6.2.2 CustomCalibrationWrapper.h

```
#pragma once

#include <cstdint>
```

```
#include <vector>

namespace CustomCalibWrapper
{
        class CustomCalibrationWrapper
        {
        private:
                void* mData;
                void *mRs400Device;
        public:
                CustomCalibrationWrapper(void *rs400Dev, int chessboardWidth, int 
chessboardHeight, float chessboardSquareSize, int numCameras, int numImages);
                ~CustomCalibrationWrapper();

                bool AddImage(uint8_t* image, int width, int height, int stride, int 
cameraIndex, int imageIndex);
                int CalculateCalibration(int lrWidth, int lrHeight, int rgbWidth, int 
rgbHeight);
        };
}
```

### 6.2.3 CalibrationManager.h

```
/*
* INTEL CORPORATION PROPRIETARY INFORMATION
* This software is supplied under the terms of a license agreement
* or nondisclosure agreement with Intel Corporation and may not be
* copied or disclosed except in accordance with the terms of that
* agreement.
* Copyright(c) 2016-2017 Intel Corporation. All Rights Reserved.
*/
#pragma once

#include <chrono>
#include "Rs400Device.h"
#include "CustomCalibrationWrapper.h"

namespace CustomCalibrator
{
        const int NUM_SHOTS = 6;
        const int CHESSBOARD_HEIGHT = 6;
        const int CHESSBOARD_WIDTH = 7;
        const float CHESSBOARD_SQUARE_SIZE = 60.0f;

        typedef struct tagCOLOR
        {
                int R;
                int G;
                int B;
                int A;
        } COLOR;

        typedef struct tagPOINT_D
        {
                double x;
                double y;
        } POINT_D;

        class Stopwatch
        {
```

```
		typedef std::chrono::high_resolution_clock clock;
		typedef std::chrono::milliseconds milliseconds;

		clock::time_point start_;

	private:
		milliseconds intervalMs(const clock::time_point& t1, const 
clock::time_point& t0)
		{
			return std::chrono::duration_cast<milliseconds>(t1 - t0);
		}

		clock::time_point now()
		{
			return clock::now();
		}

	public:
		Stopwatch() : start_(clock::now()) {}

		void Start()
		{
			start_ = clock::now();
		}

		clock::time_point Restart()
		{
			start_ = clock::now();
			return start_;
		}

		double ElapsedMilliseconds()
		{
			return 1.0 * intervalMs(now(), start_).count();
		}
	};

	class CustomCalibration
	{
	public:
		CustomCalibration();
		virtual ~CustomCalibration();

		/* Initialization */
		bool Initialize(void);

		void Start(void);

	private:
		void InitializeGL(void *parentClass);

		void OnDisplay(void);
		void OnIdle(void);
		void OnKeyBoard(unsigned char key, int x, int y);
		void OnClose(void);

		void RenderLines(std::vector<POINT_D>, COLOR color, int lineWidth);
		void RenderText(float x, float y, void *font, const char* string, COLOR 
color);
```

```
                void ConvertLuminance16ToLuminance8(const uint16_t* image, int width, int
height, uint8_t* output);
                void ConvertYUY2ToLuminance8(const uint8_t* image, int width, int height,
uint8_t* output);
                void ConvertYUY2ToRGBA(const uint8_t* image, int width, int height,
uint8_t* output);

        private:
                RsCamera::Rs400Device *m_rsDevice;
                RsCamera::camera_info m_cameraInfo;
                CustomCalibWrapper::CustomCalibrationWrapper* m_calibWrapper;

                std::unique_ptr<uint8_t[]> m_leftImage;
                std::unique_ptr<uint8_t[]> m_rightImage;
                std::unique_ptr<uint32_t[]> m_colorImage;
                std::unique_ptr<uint8_t[]> m_grayImage;

                int m_width;
                int m_height;
                int m_fps;
                int m_rgbWidth;
                int m_rgbHeight;

                bool m_imagesCapture;
                bool m_captureStarted;
                int  m_numShot;
                bool m_inProcessing;
                bool m_inUpdating;
                bool m_shotFailed;

                Stopwatch *m_watch;
                GLuint m_textures[3];
        };
}
```

### 6.2.4 DS5CalibCoefficients.h

```
/*
 * INTEL CORPORATION PROPRIETARY INFORMATION
 * This software is supplied under the terms of a license agreement
 * or nondisclosure agreement with Intel Corporation and may not be
 * copied or disclosed except in accordance with the terms of that
 * agreement.
 * Copyright(c) 2016-2017 Intel Corporation. All Rights Reserved.
 */

/** @file DS5CalibCoefficients.h */
#pragma once

#include <cstdint>
#include <vector>
#include <cstddef>

namespace dscam
{

#pragma pack(push, 1)

//! Calibration table header for DS5. Generic for all tables.
struct DS5CalibrationTableHeader
```

```
{
        uint16_t version;                         //!< Table version
        uint16_t tableType;                       //!< Table type
        uint32_t tableSize;                       //!< Table size
        uint32_t fullVersion;                     //!< Complete version of the table
        uint32_t crc32;                           //!< CRC32 of the content
};

//! DS5 calibration intrinsics parameters for non-rectified camera
struct DS5CoeffsIntrinsics
{
        float fx;                                 //!< Focal length in x axis, normalized
by width
        float fy;                                 //!< Focal length in y axis, normalized
by height
        float px;                                 //!< Principal point x axis coordinate,
normalized by width
        float py;                                 //!< Principal point y axis coordinate,
normalized by height
        float k[5];                               //!< Distortion coefficients
};

//! DS5 calibration intrinsics parameters for rectified camera
struct DS5CoeffsMode
{
        float rfx;                                //!< Rectified focal length in x axis
        float rfy;                                //!< Rectified focal length in y axis
        float rpx;                                //!< Rectified principal point x axis
coordinate
        float rpy;                                //!< Rectified principal point y axis
coordinate
};

//! DS5 calibration coefficients. Contains rectification parameters and data used by the
chip to produce depth map.
struct DS5CalibrationCoefficients
{
        DS5CoeffsIntrinsics instrinsicsLeft;      //!< Intrinsics of the left camera
        DS5CoeffsIntrinsics instrinsicsRight;     //!< Intrinsics of the right camera
        float Rleft[9];                           //!< Inverse rotation of the left camera
in rectified coordinate system
        float Rright[9];                          //!< Inverse rotation of the left camera
in rectified coordinate system
        float B;                                  //!< Baseline (maintains polarity)
        uint32_t useBrownModel;                   //!< Whether to use Brown or TYZX
distrotion model (tangential distortion)
        uint8_t  reserved[88];                    //!< Reserved
};

//! DS5 recitifed resolution index
enum DS5CoeffsModeIndex
{
        DS5M_1920_1080 = 0,                       //!< 1920x1080 rectified resolution (R8L8
only)
        DS5M_1280_720  = 1,                       //!< 1280x720 rectified resolution
        DS5M_640_480   = 2,                       //!< 640x480 rectified resolution
        DS5M_848_480   = 3,                       //!< 848x480 rectified resolution
        DS5M_640_360   = 4,                       //!< 640x360 rectified resolution
        DS5M_424_240   = 5,                       //!< 424x240 rectified resolution
        DS5M_320_240   = 6,                       //!< 320x240 rectified resolution
        DS5M_480_270   = 7,                       //!< 480x270 rectified resolution
        DS5M_1280_800  = 8,                     //!< 1280x800 rectified resolution (R8L8
only)
        DS5M_960_540   = 9                      //!< 960x540 rectified resolution
};

//! DS5 calibration type
enum DS5CalibrationType
{
        DS5CT_Custom            = 0x00,         //!< Custom calibration type (or
otherwise undefined)
        DS5CT_Production        = 0x01,         //!< Production calibration
        DS5CT_Technician        = 0x02,         //!< Technician calibration
        DS5CT_OEM               = 0x03,         //!< OEM calibration
        DS5CT_DynamicTargetless = 0x04,         //!< Dynamic calibration (targetless)
        DS5CT_DynamicWithTarget = 0x05,         //!< Dynamic calibration (with target)
        DS5CT_TechnicianNonRT   = 0x06,         //!< Technician calibration (non-RT
algorithm)
        DS5CT_Validation        = 0x07,         //!< Calibration done for validatoin
purposes

        DS5CT_CustomerFlag      = 0x1000        //!< Customer used their own calibration.
This is used as a flag and sticks after dynamic calibration.
};

//! DS5 calibration coefficients table. Table ID is 0x19.
struct DS5CalibrationCoefficientsTable
{
        DS5CalibrationTableHeader header;       //!< Header
        DS5CalibrationCoefficients coeffs;      //!< Coefficients
        DS5CoeffsMode  modes[12];               //!< Rectified camera parameters. Each
entry in the table corresponds to a resolution
        uint8_t  reserved2[64];                 //!< Reserved
};

//! Structure used to update the calibration during live streaming using the CALIBRECALC
command
struct DS5CalibrationCoefficientsUpdateParam
{
        DS5CalibrationCoefficients coeffs;      //!< Coefficients
        DS5CoeffsMode mode;                     //!< Rectified camera parameters for the
current resolution
};

//! DS5 raw calibration intrinsics parameters for non-rectified camera
struct DS5CalParameters
{
        float K[9];                             //!< Intrinsics stored as 3x3 matrix,
normalized by width and height
        float d[5];                             //!< Distortion coefficients
        float R[3];                             //!< The rotation of the camera as
Rodriquez rotation vector
        float T[3];                             //!< The translation of the camera,
passed as T = -Rt, where t is the translation vector
};

//! DS5 depth calibration table for versions 2.0 and 3.0. Contains raw calibraiton data.
Table ID = 0x1F.
struct DS5DepthCalibrationTableV2030
{
        DS5CalibrationTableHeader header;       //!< Header
        uint32_t rmax;                          //!< Maximum permissible rotation
        DS5CalParameters parametersLeft;        //!< Parameters of the left camera
        DS5CalParameters parametersRight;       //!< Parameters of the right camera
        float Rd[3];                            //!< Rotation vector of the depth camera 
against world coordinates. Stored as Rodriguez rotation vector.
        float Td[3];                            //!< The new translation between non-
rectified left and right camera, passed as T=-Rt, where t is the translation vector and R 
is the rotaion matrix
        uint16_t imageWidth;                    //!< Width of the images used during 
calibration
        uint16_t imageHeight;                   //!< Height of the images used during 
calibration
        uint8_t reserved[48];                   //!< Reserved
};

//! DS5 depth calibration table. Contains raw calibraiton data. Table ID = 0x1F.
struct DS5DepthCalibrationTable
{
        DS5CalibrationTableHeader header;       //!< Header
        uint32_t rmax;                          //!< Maximum permissible rotation
        DS5CalParameters parametersLeft;        //!< Intrinsics of the left camera
        DS5CalParameters parametersRight;       //!< Intrinsics of the right camera
        float Kd[9];                            //!< Intrinsics of the depth camera at 
calibration resolution
        float Rd[3];                            //!< Rotation vector of the depth camera 
against world coordinates. Stored as Rodriguez rotation vector.
        float Td[3];                            //!< The new translation between non-
rectified left and right camera, passed as T=-Rt, where t is the translation vector and R 
is the rotaion matrix
        uint16_t imageWidth;                    //!< Width of the images used during 
calibration
        uint16_t imageHeight;                   //!< Height of the images used during 
calibration
        uint16_t calibrationType;               //!< Calibration type
        uint8_t reserved[10];                   //!< Reserved
};

//! DS5 RGB calibration table. Contains RGB calibraiton data. Table ID = 0x20.
struct DS5RGBCalibrationTable
{
        DS5CalibrationTableHeader header;       //!< Header
        DS5CalParameters parameters;            //!< Intrinsics of the RGB camera
        float P[12];                            //!< Projection matrix from depth to 
unrectified RGB camera
        uint16_t imageWidth;                    //!< Width of the images used during 
calibration
        uint16_t imageHeight;                   //!< Height of the images used during 
calibration
        float Krect[9];                         //!< Intrinsics of the rectified RGB 
camera
        float Rrect[9];                         //!< Inverse rotation of the RGB camera 
in rectified coordinate system
        float Trect[3];                         //!< Translation of the RGB camera in 
rectified coordinate system
        uint8_t reserved[24];                   //!< Reserved
};

//! DS5 table creation status
enum DS5TableCreationStatus
{
        DS5CTS_OK,                              //!< OK
        DS5CTS_InvalidTable,                    //!< One of the tables was invalid
        DS5CTS_InvalidLeftRightResolution,      //!< Left & Right camera resolution is
not supported
        DS5CTS_LeftIntrinsicsUnreasonable,      //!< Left camera intrinsics unreasonable
        DS5CTS_RightIntrinsicsUnreasonable      //!< Right camera intrinsics unreasonable
};

//! DS5 table parse status
enum DS5TableParseStatus
{
        DS5CPS_OK,                              //!< OK
        DS5CPS_InvalidTable,                    //!< Some of the tables were not valid or
correct version
        DS5CPS_InvalidParameter                 //!< One of the output variables was not
valid
};

#pragma pack(pop)

#if 0
/** Makes sure that the table are the latest version (upgrades them if not). Additional
elements in the calibration table
 *  that cannot be recovered if the table is old will be set to 0, thus make sure that
the version is noted properly before
 *  calling this function for any disabilities.
 *
 *  @param coeffsTable: The coefficients table in any supported version, which will be
converted to latest version.
 *  @param depthTable: The depth table in any supported version, which will be converted
to latest version.
 *  @param rgbTable: The RGB table in any supported version, which will be converted to
latest version. This parameter is optional.
 *  @return Will fail if the version is too old.
 */
bool EnsureTablesAreTheLatest(void* coeffsTable, void* depthTable, void* rgbTable =
NULL);

/** Calculates CRC32
 *
 *  @param buffer: The data
 *  @param length: Length of the data
 *  @return CRC32 checksum
 */
uint32_t CalculateCrc32(uint8_t *buffer, int length);

/** Updates CRC32 checksum in calibration coefficients table
 *
 *  @param coeffsTable: The calibration coefficients table
 */
void UpdateCrc32(DS5CalibrationCoefficientsTable* coeffsTable);

/** Updates CRC32 checksum in depth calibration table
 *
 *  @param depthTable: The depth calibration table
 */
void UpdateCrc32(DS5DepthCalibrationTable* depthTable);

/** Updates CRC32 checksum in RGB calibration table
 *
 *  @param rgbTable: The RGB calibration table
 */
void UpdateCrc32(DS5RGBCalibrationTable* rgbTable);
```

```
/** Update the calibration tables with recalculated extrinsic parameters.
 *
 *  @param coeffsTable: The calibration coefficients table
 *  @param depthTable: The depth calibration table
 *  @param rgbTable: The RGB calibration table - this parameter is optional and may be
NULL
 *  @param calibrationType: Method used for the re-calibration
 *  @param R: The new rotation between non-rectified left and right camera, passed as 3x3
rotation matrix
 *  @param T: The new translation between non-rectified left and right camera, passed as
T=-Rt, where t is the translation vector
 *  @return Whether the calibration recalculation succeeded
 */
bool UpdateCalibration(DS5CalibrationCoefficientsTable* coeffsTable,
DS5DepthCalibrationTable* depthTable, DS5RGBCalibrationTable* rgbTable,
DS5CalibrationType calibrationType, double R[9], double T[3]);

/** <B>INTERNAL USE ONLY!</B><BR>
 *  Calculates updated rectification parameters for specified resolution.
 *
 *  @param coeffsTable: The calibration coefficients table
 *  @param depthTable: The depth calibration table
 *  @param R: The new rotation between non-rectified left and right camera, passed as 3x3
rotation matrix
 *  @param T: The new translation between non-rectified left and right camera, passed as
T=-Rt, where t is the translation vector
 *  @param modeWidth: Image width of the rectified camera
 *  @param modeHeight: Image height of the rectified camera
 *  @param mode: The parameter will be populated with the rectified camera intrinsics
 *  @param Rleft: The inverse rotation of the left camera in the new rectified coordinate
system
 *  @param Rright: The inverse rotation of the right camera in the new rectified
coordinate system
 *  @return Whether the calibration recalculation succeeded
 */
bool RecalculateMode(const DS5CalibrationCoefficientsTable& coeffsTable, const
DS5DepthCalibrationTable& depthTable, double R[9], double T[3], int modeWidth, int
modeHeight, DS5CoeffsMode& mode, double Rleft[9], double Rright[9]);

/** Creates calibration tables based on supplied parameters, which assumes that left
camera is the reference camera and is located at world origin.
 *
 *  @param coeffsTable: The calibration coefficients table
 *  @param depthTable: The depth calibration table
 *  @param rgbTable: The RGB calibration table - this parameter is optional and may be
NULL
 *  @param resolutionLeftRight: The resolution of the left and right camera, specified as
[width; height]
 *  @param focalLengthLeft: The focal length of the left camera, specified as [fx; fy] in
pixels
 *  @param principalPointLeft: The principal point of the left camera, specified as [px;
py] in pixels
 *  @param distortionLeft: The distortion of the left camera, specified as Brown's
distortion model [k1; k2; p1; p2; k3]
 *  @param focalLengthRight: The focal length of the right camera, specified as [fx; fy]
in pixels
 *  @param principalPointRight: The principal point of the right camera, specified as
[px; py] in pixels
 *  @param distortionRight: The distortion of the right camera, specified as Brown's
distortion model [k1; k2; p1; p2; k3]
```

```
 *  @param rotationLeftRight: The rotation from the right camera coordinate system to the
left camera coordinate system, specified as a 3x3 row-major rotation matrix
 *  @param translationLeftRight: The translation from the right camera coordinate system
to the left camera coordinate system, specified as a 3x1 vector in milimeters
 *  @param hasRGB: Whether RGB camera calibration parameters are supplied
 *  @param resolutionRGB: The resolution of the RGB camera, specified as [width; height]
 *  @param focalLengthRGB: The focal length of the RGB camera, specified as [fx; fy] in
pixels
 *  @param principalPointRGB: The principal point of the RGB camera, specified as [px;
py] in pixels
 *  @param distortionRGB: The distortion of the RGB camera, specified as Brown's
distortion model [k1; k2; p1; p2; k3]
 *  @param rotationLeftRGB: The rotation from the RGB camera coordinate system to the
left camera coordinate system, specified as a 3x3 row-major rotation matrix
 *  @param translationLeftRGB: The translation from the RGB camera coordinate system to
the left camera coordinate system, specified as a 3x1 vector in milimeters
 *  @return When DS5CTS_OK then the tables were created successfully
 */
DS5TableCreationStatus CreateCalibrationTables(DS5CalibrationCoefficientsTable*
coeffsTable, DS5DepthCalibrationTable* depthTable, DS5RGBCalibrationTable* rgbTable,
const int resolutionLeftRight[2], const double focalLengthLeft[2], const double
principalPointLeft[2],
        const double distortionLeft[5], const double focalLengthRight[2], const double
principalPointRight[2], const double distortionRight[5], const double rotationRight[9],
const double translationRight[3], const bool hasRGB, const int resolutionRGB[2],
        const double focalLengthRGB[2], const double principalPointRGB[2], const double
distortionRGB[5], const double rotationRGB[9], const double translationRGB[3]);

/** Parses calibration tables into ordinary calibraiton parameters
 *
 *  @param coeffsTable: The calibration coefficients table
 *  @param depthTable: The depth calibration table
 *  @param rgbTable: The RGB calibration table - this parameter is optional and may be
NULL
 *  @param resolutionLeftRight: The resolution of the left and right camera, specified as
[width; height]
 *  @param focalLengthLeft: The focal length of the left camera, specified as [fx; fy] in
pixels
 *  @param principalPointLeft: The principal point of the left camera, specified as [px;
py] in pixels
 *  @param distortionLeft: The distortion of the left camera, specified as Brown's
distortion model [k1; k2; p1; p2; k3]
 *  @param focalLengthRight: The focal length of the right camera, specified as [fx; fy]
in pixels
 *  @param principalPointRight: The principal point of the right camera, specified as
[px; py] in pixels
 *  @param distortionRight: The distortion of the right camera, specified as Brown's
distortion model [k1; k2; p1; p2; k3]
 *  @param rotationLeftRight: The rotation from the right camera coordinate system to the
left camera coordinate system, specified as a 3x3 rotation matrix
 *  @param translationLeftRight: The translation from the right camera coordinate system
to the left camera coordinate system, specified as a 3x1 vector in milimeters
 *  @param hasRGB: Whether RGB camera calibration parameters are supplied
 *  @param resolutionRGB: The resolution of the RGB camera, specified as [width; height]
 *  @param focalLengthRGB: The focal length of the RGB camera, specified as [fx; fy] in
pixels
 *  @param principalPointRGB: The principal point of the RGB camera, specified as [px;
py] in pixels
 *  @param distortionRGB: The distortion of the RGB camera, specified as Brown's
distortion model [k1; k2; p1; p2; k3]
```

```
 *  @param rotationLeftRGB: The rotation from the RGB camera coordinate system to the
left camera coordinate system, specified as a 3x3 rotation matrix
 *  @param translationLeftRGB: The translation from the RGB camera coordinate system to
the left camera coordinate system, specified as a 3x1 vector in milimeters
 *  @return When DS5CPS_OK then the tables were parsed successfully
 */
DS5TableParseStatus ParseCalibrationTables(const DS5CalibrationCoefficientsTable*
coeffsTable, const DS5DepthCalibrationTable* depthTable, const DS5RGBCalibrationTable*
rgbTable, int resolutionLeftRight[2], double focalLengthLeft[2], double
principalPointLeft[2],
        double distortionLeft[5], double focalLengthRight[2], double
principalPointRight[2], double distortionRight[5], double rotationRight[9], double
translationRight[3], bool& hasRGB, int resolutionRGB[2], double focalLengthRGB[2], double
principalPointRGB[2],
        double distortionRGB[5], double rotationRGB[9], double translationRGB[3]);
#endif
}
```

## 6.2.5 Rs400Device.h

```
/*
* INTEL CORPORATION PROPRIETARY INFORMATION
* This software is supplied under the terms of a license agreement
* or nondisclosure agreement with Intel Corporation and may not be
* copied or disclosed except in accordance with the terms of that
* agreement.
* Copyright(c) 2016-2017 Intel Corporation. All Rights Reserved.
*/
#pragma once

#ifdef _WIN32
#include <Windows.h>
#else
#include <pthread.h>
#endif

#include <iostream>
#include <sstream>
#include <string>
#include <vector>

#include <functional>
#include <memory>       // For shared_ptr
#include <thread>

#include "rs.hpp"

#ifdef _WIN32
#define MUTEX_LOCK      EnterCriticalSection
#define MUTEX_UNLOCK    LeaveCriticalSection
#else
#define MUTEX_LOCK(m)    if (0 != pthread_mutex_lock(m)) throw std::runtime_error
("pthread_mutex_lock failed")
#define MUTEX_UNLOCK(m)  if (0 != pthread_mutex_unlock(m)) throw std::runtime_error
("pthread_mutex_unlock failed")
#endif

namespace RsCamera
{
#define NELEMS(x)  (sizeof(x) / sizeof((x)[0]))
```

```
    enum RS_400_STREAM_TYPE
    {
        RS400_STREAM_INFRARED,
        RS400_STREAM_INFRARED2,
        RS400_STREAM_COLOR,
        RS400_STREAM_COUNT
    };

    enum GVD_FIELDS
    {
        SKU_COMPONENT = 166,
        RGB_MODE = 178,
    };

        struct camera_info
        {
                std::string name;
                std::string serial;
                std::string fw_ver;
                std::string pid;
                bool isRGB;
                bool isWide;
        };

    class Rs400Device
    {
    public:
        Rs400Device();
        virtual ~Rs400Device();

                camera_info InitializeCamera();
        void *GetDeviceHandle() { return (void *)&m_device; }

        bool SetMediaMode(int width, int height, int frameRate, int colorWidth, int 
colorHeight, bool enableColor);

        //If depth stream is enabled, otherImage will be depth image,
        //otherwise it will be right image
        void StartCapture(std::function<void(const void *leftImage, const void 
*colorImage,
            const void *rgbImage, const uint64_t timeStamp)> callback);
        void StopCapture();

        void EnableAutoExposure(float value);
                void EnableEmitter(float value);
                void SetAeControl(unsigned int point);

    private:
        bool GetProfile(rs2::stream_profile& profile, rs2_stream stream, int width, int 
height,
            int fps, int index);

    private:
        rs2::context *m_context;
        rs2::device m_device;
        rs2::sensor m_depthSensor;
                rs2::sensor m_colorSensor;

        std::function<void(const void *leftImage, const void *rightImage, const void 
*colorImage,
            const uint64_t timeStamp)> m_callback;

        std::vector<rs2::stream_profile> m_depthProfiles;
                std::vector<rs2::stream_profile> m_colorProfile;
        std::unique_ptr<uint16_t[]> m_lrImage[2];

        void *m_pData[RS400_STREAM_COUNT + 1];
        uint64_t m_timestamp[RS400_STREAM_COUNT + 1];
        uint64_t m_ts;

#ifdef _WIN32
        CRITICAL_SECTION m_mutex;
#else
        pthread_mutex_t  m_mutex;
#endif
        bool m_stopProcessFrame;
        bool m_captureStarted;
        bool m_bColorEnabled;
    };
}
```

## 6.3 C++ Files

### 6.3.1 Main.cpp

```
/*
* INTEL CORPORATION PROPRIETARY INFORMATION
* This software is supplied under the terms of a license agreement
* or nondisclosure agreement with Intel Corporation and may not be
* copied or disclosed except in accordance with the terms of that
* agreement.
* Copyright(c) 2016-2017 Intel Corporation. All Rights Reserved.
*/

#include "GL/freeglut.h"

#include "CalibrationManager.h"

using namespace CustomCalibrator;

int main(int argc, char * argv[])
{
        CustomCalibration *customCalib = new CustomCalibration();
        if (!customCalib->Initialize()) return EXIT_FAILURE;

        customCalib->Start();
        glutMainLoop();

        return EXIT_SUCCESS;
}
```

### 6.3.2 CustomCalibration.cpp

```
/*
 * INTEL CORPORATION PROPRIETARY INFORMATION
 * This software is supplied under the terms of a license agreement
 * or nondisclosure agreement with Intel Corporation and may not be
 * copied or disclosed except in accordance with the terms of that
 * agreement.
 * Copyright(c) 2016-2017 Intel Corporation. All Rights Reserved.
 */

#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/calib3d.hpp>
#include <vector>

#include "CustomCalibration.h"

using namespace std;
using namespace cv;

bool DetectChessboard(const Mat& image, const Size& chessboardSize, vector<Point2f>&
corners)
{
        // Find chessboard corners
        if (!findChessboardCorners(image, chessboardSize, corners,
CALIB_CB_ADAPTIVE_THRESH | CALIB_CB_NORMALIZE_IMAGE | CALIB_CB_FILTER_QUADS))
                return false;

        // Refine them
        cornerSubPix(image, corners, Size(11, 11), Size(-1, -1),
TermCriteria(CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
        return true;
}

static void CreateCorners3D(const Size& chessboardSize, float checkerSize, size_t
numImages, vector<vector<Point3f> >& corners3D)
{
        corners3D.resize(numImages);
        corners3D[0].resize(chessboardSize.width * chessboardSize.height);

        for (int i = 0; i < chessboardSize.height; i++)
                for (int j = 0; j < chessboardSize.width; j++)
                        corners3D[0][i * chessboardSize.width + j] = Point3f(j *
checkerSize, i * checkerSize, 0.0f);

        for (size_t i = 1; i < numImages; i++)
                        corners3D[i] = corners3D[0];
}

double CalibrateDepthCamera(const vector<vector<Point2f> >& cornersLeft, const
vector<vector<Point2f> >& cornersRight, const Size& chessboardSize, float checkerSize,
const Size& imageSizeLR, Mat& Kl, Mat& Dl, Mat& Kr, Mat& Dr, Mat& Rlr, Mat& Tlr)
{
        CV_Assert(cornersLeft.size() != 0 && cornersLeft.size() == cornersRight.size());
        CV_Assert(checkerSize > 0.0f);

        // Create 3D prototype of the corners
        vector<vector<Point3f> > corners3D;
        CreateCorners3D(chessboardSize, checkerSize, cornersLeft.size(), corners3D);

        // Calibrate each camera individualy
        calibrateCamera(corners3D, cornersLeft, imageSizeLR, Kl, Dl, noArray(),
noArray(), CV_CALIB_FIX_ASPECT_RATIO, TermCriteria(TermCriteria::COUNT +
TermCriteria::EPS, 60, DBL_EPSILON));
```

```
        calibrateCamera(corners3D, cornersRight, imageSizeLR, Kr, Dr, noArray(),
noArray(), CV_CALIB_FIX_ASPECT_RATIO, TermCriteria(TermCriteria::COUNT +
TermCriteria::EPS, 60, DBL_EPSILON));

        // Calibrate the extrinsics between them
        return stereoCalibrate(corners3D, cornersLeft, cornersRight, Kl, Dl, Kr, Dr,
Size(-1, -1), Rlr, Tlr, noArray(), noArray(), CV_CALIB_FIX_INTRINSIC |
CV_CALIB_USE_INTRINSIC_GUESS, TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 300,
DBL_EPSILON));
}

double CalibrateRGBCamera(const vector<vector<Point2f> >& cornersLeft, const
vector<vector<Point2f> >& cornersRGB, const Size& chessboardSize, float checkerSize,
const Size& imageSizeRGB, const Mat& Kl, const Mat& Dl, Mat& Kc, Mat& Dc, Mat& Rlc, Mat&
Tlc)
{
        CV_Assert(cornersLeft.size() != 0 && cornersLeft.size() == cornersRGB.size());
        CV_Assert(checkerSize > 0.0f);

        // Create 3D prototype of the corners
        vector<vector<Point3f> > corners3D;
        CreateCorners3D(chessboardSize, checkerSize, cornersLeft.size(), corners3D);

        // Calibrate RGB camera
        calibrateCamera(corners3D, cornersRGB, imageSizeRGB, Kc, Dc, noArray(),
noArray(), CV_CALIB_FIX_ASPECT_RATIO, TermCriteria(TermCriteria::COUNT +
TermCriteria::EPS, 60, DBL_EPSILON));

        // Calibrate the extrinsics between them
        return stereoCalibrate(corners3D, cornersLeft, cornersRGB, Kl, Dl, Kc, Dc, Size(-
1, -1), Rlc, Tlc, noArray(), noArray(), CV_CALIB_FIX_INTRINSIC |
CV_CALIB_USE_INTRINSIC_GUESS, TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 300,
DBL_EPSILON));
}
```

### 6.3.3 CustomCalibrationWrapper.cpp

```
#include "DS5CalibCoefficients.h"
#include "DSDynamicCalibration.h"
#include "CustomCalibration.h"
#include "CustomCalibrationWrapper.h"
#include <vector>

using namespace std;
using namespace cv;
using namespace dscam;

#define mCustomCalibratorData ((CustomCalibratorData*)mData)
#define ccwhandle ((CustomCalibrationWrapper*)handle)

namespace CustomCalibWrapper
{

struct CustomCalibratorData
{
        vector<vector<vector<Point2f> > > corners;
        Size chessboardSize;
        float chessboardSquareSize;
};
```

```
CustomCalibrationWrapper::CustomCalibrationWrapper(void *rs400Dev, int chessboardWidth,
int chessboardHeight, float chessboardSquareSize, int numCameras, int numImages)
{
        mRs400Device = rs400Dev;
        mData = new CustomCalibratorData();

        mCustomCalibratorData->chessboardSize = Size(chessboardWidth, chessboardHeight);
        mCustomCalibratorData->chessboardSquareSize = chessboardSquareSize;

        mCustomCalibratorData->corners.resize(numCameras);
        for (int i = 0; i < numCameras; i++)
                mCustomCalibratorData->corners[i].resize(numImages);
}

CustomCalibrationWrapper::~CustomCalibrationWrapper()
{
        if (mData)
        {
                delete mData;
                mData = NULL;
        }
}

bool CustomCalibrationWrapper::AddImage(uint8_t* image, int width, int height, int
stride, int cameraIndex, int imageIndex)
{
        Mat imgMat(Size(width, height), CV_8U, image, stride);
        return DetectChessboard(imgMat, mCustomCalibratorData->chessboardSize,
mCustomCalibratorData->corners[cameraIndex][imageIndex]);
}

int CustomCalibrationWrapper::CalculateCalibration(int lrWidth, int lrHeight, int
rgbWidth, int rgbHeight)
{
        try
        {
                bool hasRGB = mCustomCalibratorData->corners.size() > 2;

                Mat Kl, Dl, Kr, Dr, Rlr, Tlr;
                CalibrateDepthCamera(mCustomCalibratorData->corners[0],
mCustomCalibratorData->corners[1], mCustomCalibratorData->chessboardSize,
mCustomCalibratorData->chessboardSquareSize, Size(lrWidth, lrHeight), Kl, Dl, Kr, Dr,
Rlr, Tlr);

                Mat Kc, Dc, Rlc, Tlc;
                if (hasRGB)
                        CalibrateRGBCamera(mCustomCalibratorData->corners[0],
mCustomCalibratorData->corners[2], mCustomCalibratorData->chessboardSize,
mCustomCalibratorData->chessboardSquareSize, Size(rgbWidth, rgbHeight), Kl, Dl, Kc, Dc,
Rlc, Tlc);
                else
                {
                        Kc = Mat::zeros(3, 3, CV_64F);
                        Dc = Mat::zeros(1, 5, CV_64F);
                        Rlc = Mat::zeros(3, 3, CV_64F);
                        Tlc = Mat::zeros(3, 1, CV_64F);
                }

                int resolutionLR[2] = { lrWidth, lrHeight };
                double focalLengthLeft[2] = { Kl.at<double>(0,0), Kl.at<double>(1,1) };
                double principalPointLeft[2] = { Kl.at<double>(0,2), Kl.at<double>(1,2)
};
                double distortionLeft[5] = { Dl.at<double>(0,0), Dl.at<double>(0,1),
Dl.at<double>(0,2), Dl.at<double>(0,3), Dl.at<double>(0,4) };
                double focalLengthRight[2] = { Kr.at<double>(0,0), Kr.at<double>(1,1) };
                double principalPointRight[2] = { Kr.at<double>(0,2), Kr.at<double>(1,2)
};
                double distortionRight[5] = { Dr.at<double>(0,0), Dr.at<double>(0,1),
Dr.at<double>(0,2), Dr.at<double>(0,3), Dr.at<double>(0,4) };

                double rotationLR[9] = { Rlr.at<double>(0,0), Rlr.at<double>(0,1),
Rlr.at<double>(0,2), Rlr.at<double>(1,0), Rlr.at<double>(1,1),
                        Rlr.at<double>(1,2), Rlr.at<double>(2,0), Rlr.at<double>(2,1),
Rlr.at<double>(2,2) };
                double translationLR[3] = { Tlr.at<double>(0,0), Tlr.at<double>(1,0),
Tlr.at<double>(2,0) };

                int resolutionRGB[2] = { rgbWidth, rgbHeight };
                double focalLengthRGB[2] = { Kc.at<double>(0,0), Kc.at<double>(1,1) };
                double principalPointRGB[2] = { Kc.at<double>(0,2), Kc.at<double>(1,2) };
                double distortionRGB[5] = { Dc.at<double>(0,0), Dc.at<double>(0,1),
Dc.at<double>(0,2), Dc.at<double>(0,3), Dc.at<double>(0,4) };

                double rotationLC[9] = { Rlc.at<double>(0,0), Rlc.at<double>(0,1),
Rlc.at<double>(0,2), Rlc.at<double>(1,0), Rlc.at<double>(1,1),
                        Rlc.at<double>(1,2), Rlc.at<double>(2,0), Rlc.at<double>(2,1),
Rlc.at<double>(2,2) };
                double translationLC[3] = { Tlc.at<double>(0,0), Tlc.at<double>(1,0),
Tlc.at<double>(2,0) };

                DynamicCalibrationAPI::DSDynamicCalibration* dyCalib = new
DynamicCalibrationAPI::DSDynamicCalibration();
                dyCalib->Initialize(mRs400Device,
DynamicCalibrationAPI::DSDynamicCalibration::CalibrationMode::CAL_MODE_USER_CUSTOM,
lrWidth, lrHeight);
                int status = dyCalib->WriteCustomCalibrationParameters(resolutionLR,
focalLengthLeft, principalPointLeft, distortionLeft, focalLengthRight,
                        principalPointRight, distortionRight, rotationLR, translationLR,
hasRGB, resolutionRGB, focalLengthRGB, principalPointRGB,
                        distortionRGB, rotationLC, translationLC);

                return status;
        }
        catch (...)
        {
                return -1;
        }
}

}
```

## 6.3.4 CalibrationManager.cpp

```
/*
* INTEL CORPORATION PROPRIETARY INFORMATION
* This software is supplied under the terms of a license agreement
* or nondisclosure agreement with Intel Corporation and may not be
* copied or disclosed except in accordance with the terms of that
* agreement.
* Copyright(c) 2016-2017 Intel Corporation. All Rights Reserved.
*/
#pragma once

#include <iostream>
#include <string>
#include "GL/freeglut.h"
#include "CalibrationManager.h"
#include "CustomCalibration.h"

using namespace std;
using namespace CustomCalibrator;
using namespace RsCamera;
using namespace CustomCalibWrapper;

CustomCalibration::CustomCalibration()
{
        m_rsDevice = new Rs400Device();
        m_captureStarted = false;
        m_numShot = 0;
        m_imagesCapture = false;
        m_inProcessing = false;
        m_inUpdating = false;
        m_shotFailed = false;

        m_watch = new Stopwatch();
}

CustomCalibration::~CustomCalibration()
{
        if (m_captureStarted)
                m_rsDevice->StopCapture();
}

bool CustomCalibration::Initialize()
{
        int numCameras = 2;

        InitializeGL((void *)this);

        m_cameraInfo = m_rsDevice->InitializeCamera();

        if (m_cameraInfo.name.empty()) return false;

        if (m_cameraInfo.isWide)
        {
                m_width = 1280;
                m_height = 800;
        }
        else
        {
                m_width = 1920;
                m_height = 1080;
        }

        m_fps = 15;

        m_rgbWidth = 1920;
        m_rgbHeight = 1080;

        m_leftImage = std::unique_ptr<uint8_t[]>(new uint8_t[m_width*(m_height + 1)]);
        m_rightImage = std::unique_ptr<uint8_t[]>(new uint8_t[m_width*(m_height + 1)]);
        if (m_cameraInfo.isRGB)
        {
                m_colorImage = std::unique_ptr<uint32_t[]>(new
uint32_t[m_rgbWidth*(m_rgbHeight + 1)]);
                m_grayImage  = std::unique_ptr<uint8_t[]>(new
uint8_t[m_rgbWidth*(m_rgbHeight + 1)]);
                numCameras += 1;
        }

        m_calibWrapper = new CustomCalibrationWrapper(m_rsDevice->GetDeviceHandle(),
                CHESSBOARD_WIDTH, CHESSBOARD_HEIGHT, CHESSBOARD_SQUARE_SIZE, numCameras,
NUM_SHOTS);

        return true;
}

void CustomCalibration::InitializeGL(void *parentClass)
{
        int c = 0;

        // Open a GLUT window to display point cloud
        glutInit(&c, nullptr);
        glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
        glutInitWindowSize(1280, 720);
        glutCreateWindow("Intel RealSense Custom Calibrator");

        glutSetWindowData(parentClass);
        glutDisplayFunc([]() {
                CustomCalibration *pCalib = reinterpret_cast<CustomCalibration
*>(glutGetWindowData());
                pCalib->OnDisplay();
        });
        glutIdleFunc([]() {
                CustomCalibration *pCalib = reinterpret_cast<CustomCalibration
*>(glutGetWindowData());
                pCalib->OnIdle();
        });
        glutReshapeFunc([](int width, int height) {
                glViewport(0, 0, width, height);
        });
        glutCloseFunc([]() {
                CustomCalibration *pCalib = reinterpret_cast<CustomCalibration
*>(glutGetWindowData());
                pCalib->OnClose();
        });
        glutKeyboardFunc([](unsigned char key, int x, int y) {
                CustomCalibration *pCalib = reinterpret_cast<CustomCalibration
*>(glutGetWindowData());
                pCalib->OnKeyBoard(key, x, y);
        });

        glutSetOption(GLUT_ACTION_ON_WINDOW_CLOSE, GLUT_ACTION_GLUTMAINLOOP_RETURNS);


        int winWidth = glutGet(GLUT_WINDOW_WIDTH);
        int winHeight = glutGet(GLUT_WINDOW_HEIGHT);
        glViewport(0, 0, winWidth, winHeight);

        glGenTextures(3, m_textures);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glPointSize(3.0);

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}

void CustomCalibration::Start(void)
{
        if (m_captureStarted) return;

        m_captureStarted = true;
        m_rsDevice->EnableEmitter(0.0f);
        m_rsDevice->EnableAutoExposure(1.0f);
        if (m_cameraInfo.isWide)
                m_rsDevice->SetAeControl(800);
        else
                m_rsDevice->SetAeControl(500);

        m_rsDevice->SetMediaMode(m_width, m_height, m_fps, m_rgbWidth, m_rgbHeight,
m_cameraInfo.isRGB);

        m_rsDevice->StartCapture([&](const void *leftImage, const void *rightImage,
                const void *colorImage, const uint64_t timeStamp)
        {
                uint8_t *left = m_leftImage.get();
                ConvertLuminance16ToLuminance8((uint16_t *)leftImage, m_width, m_height,
left);
                uint8_t *right = m_rightImage.get();
                ConvertLuminance16ToLuminance8((uint16_t *)rightImage, m_width, m_height,
right);
                if (colorImage != nullptr)
                {
                        uint32_t *color = m_colorImage.get();
                        ConvertYUY2ToRGBA((uint8_t *)colorImage, m_rgbWidth, m_rgbHeight,
(uint8_t*)color);
                }

                if (m_imagesCapture && m_numShot < NUM_SHOTS)
                {
                        m_imagesCapture = false;

                        m_inProcessing = true;
                        glutPostRedisplay();

                        bool rt = m_calibWrapper->AddImage(left, m_width, m_height,
m_width, 0, m_numShot);
                        if (rt)
                        {
                                rt = m_calibWrapper->AddImage(right, m_width, m_height,
m_width, 1, m_numShot);
                                if (rt && m_cameraInfo.isRGB)
                                {
                                        uint8_t *gray = m_grayImage.get();
                                        ConvertYUY2ToLuminance8((uint8_t *)colorImage,
m_rgbWidth, m_rgbHeight, gray);
                                        rt = m_calibWrapper->AddImage(gray, m_rgbWidth,
m_rgbHeight, m_rgbWidth, 2, m_numShot);
                                }
                        }

                        m_inProcessing = false;
                        if (rt)
                        {
                                m_numShot += 1;
                        }
                        else
                        {
                                m_watch->Restart();
                                m_shotFailed = true;
                        }

                        if (m_numShot == NUM_SHOTS)
                        {
                                m_inUpdating = true;
                                glutPostRedisplay();

                                rt = m_calibWrapper->CalculateCalibration(m_width,
m_height, m_rgbWidth, m_rgbHeight);
                                if (rt == 0)
                    printf("Calibration successed.\n");
                                else
                    printf("Calibration failed.\n");

                                m_inUpdating = false;
                                glutLeaveMainLoop();
                        }
                }

                glutPostRedisplay();
        });
}

void CustomCalibration::ConvertLuminance16ToLuminance8(const uint16_t* image, int width,
int height, uint8_t* output)
{
        auto ptr = output;
        for (int i = 0; i < height; i++)
        {
                for (int j = 0; j < width; j++)
                {
                        uint8_t val = (uint8_t)(image[i * width + j] >> 8);
                        *ptr++ = val;
                }
        }
}

void CustomCalibration::ConvertYUY2ToLuminance8(const uint8_t* image, int width, int
height, uint8_t* output)
{
        auto out = output;
        auto in = image;
        for (int i = 0; i < height; i++)
        {
                for (int j = 0; j < width; j += 2)
                {
                        *out++ = in[0];
                        *out++ = in[2];
                        in += 4;
                }
        }
}

void CustomCalibration::OnDisplay(void)
{
        // Render image
        glEnable(GL_TEXTURE_2D);
        glColor4ub(255, 255, 255, 255);

        if (m_cameraInfo.isRGB)
        {
                glBindTexture(GL_TEXTURE_2D, m_textures[2]);
                glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
                glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
                glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
                glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);

                uint8_t *color = (uint8_t*)m_colorImage.get();
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, m_rgbWidth, m_rgbHeight, 0,
GL_RGBA, GL_UNSIGNED_BYTE, color);

                glBegin(GL_QUADS);
                glTexCoord2d(0, 1); glVertex2d(-1.0, -1.0);
                glTexCoord2d(0, 0); glVertex2d(-1.0, 1.0);
                glTexCoord2d(1, 0); glVertex2d(1.0, 1.0);
                glTexCoord2d(1, 1); glVertex2d(1.0, -1.0);
                glEnd();
        }

        glBindTexture(GL_TEXTURE_2D, m_textures[0]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);

        uint8_t *left = m_leftImage.get();
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, m_width, m_height, 0, GL_LUMINANCE,
GL_UNSIGNED_BYTE, left);

        glBegin(GL_QUADS);
        glTexCoord2d(0, 1); glVertex2d(-1.0, -1.0);
        glTexCoord2d(0, 0); glVertex2d(-1.0, -0.5);
        glTexCoord2d(1, 0); glVertex2d(-0.5, -0.5);
        glTexCoord2d(1, 1); glVertex2d(-0.5, -1.0);
        glEnd();

        glBindTexture(GL_TEXTURE_2D, m_textures[1]);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);

        uint8_t *right = m_rightImage.get();
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, m_width, m_height, 0, GL_LUMINANCE,
GL_UNSIGNED_BYTE, right);
        glBegin(GL_QUADS);
        glTexCoord2d(0, 1); glVertex2d(-0.5, -1.0);
        glTexCoord2d(0, 0); glVertex2d(-0.5, -0.5);
        glTexCoord2d(1, 0); glVertex2d(0, -0.5);
        glTexCoord2d(1, 1); glVertex2d(0, -1.0);
        glEnd();
        glDisable(GL_TEXTURE_2D);

        COLOR color = { 0xFF, 0x45, 0, 0xFF };
        if (m_inProcessing || m_inUpdating || m_shotFailed)
        {
                if (m_inProcessing || m_inUpdating)
                {
                        color.G = 0xff;
                        color.R = 0;
                        if (m_inProcessing)
                                RenderText(-5, 0, GLUT_BITMAP_TIMES_ROMAN_24, "Processing
image ...", color);
                        else
                                RenderText(-5, 0, GLUT_BITMAP_TIMES_ROMAN_24, "Processing
calibration ...", color);
                }
                else
                {
                        RenderText(-5, 0, GLUT_BITMAP_TIMES_ROMAN_24, "Corner detection
failed. Please retake the shot", color);

            if (m_watch->ElapsedMilliseconds() > 2000)
                        {
                                m_shotFailed = false;
                        }
                }
        }
        else
        {
                std::vector<POINT_D> points;
                points.push_back({ -1.0, 0.0 });
                points.push_back({ 1.0, 0.0 });

                points.push_back({ 0.0, 1.0 });
                points.push_back({ 0.0, -1.0 });

                points.push_back({ -1.0, -0.75 });
                points.push_back({ -0.502, -0.75 });

                points.push_back({ -0.498, -0.75 });
                points.push_back({ 0.0, -0.75 });

                points.push_back({ -0.75, -0.5 });
                points.push_back({ -0.75, -1.0 });

                points.push_back({ -0.25, -0.5 });
                points.push_back({ -0.25, -1.0 });

                RenderLines(points, color, 2);
        }

        color.G = 0xFF;
        color.B = 0xFF;
        string numCaptured = to_string(m_numShot) + " / " + to_string(NUM_SHOTS);
        RenderText(-3, -0.92, GLUT_BITMAP_TIMES_ROMAN_24, numCaptured.c_str(), color);

        RenderText(-2, 0.85, GLUT_BITMAP_TIMES_ROMAN_24, m_cameraInfo.name.c_str(),
color);
        string fwVersion = "FW Version: " + m_cameraInfo.fw_ver;
        RenderText(-3, 0.85, GLUT_BITMAP_TIMES_ROMAN_24, fwVersion.c_str(), color);
        string serial = "Serial Number: " + m_cameraInfo.serial;
        RenderText(-3, 0.75, GLUT_BITMAP_TIMES_ROMAN_24, serial.c_str(), color);

        glutSwapBuffers();
}

void CustomCalibration::OnIdle(void)
{
}

void CustomCalibration::OnKeyBoard(unsigned char key, int x, int y)
{
        switch (tolower(key))
        {
        case 'q':
        case 27:   //ESC
                if (m_captureStarted)
                {
                        m_rsDevice->StopCapture();
                }
                glutLeaveMainLoop();
                break;
        case 13:  //Enter
                m_imagesCapture = true;
                break;
        }
}

void CustomCalibration::OnClose(void)
{
        if (m_captureStarted)
                m_rsDevice->StopCapture();

        m_captureStarted = false;
}

void CustomCalibration::ConvertYUY2ToRGBA(const uint8_t* image, int width, int height,
uint8_t* output)
{
        int n = width*height;
        auto src = image;
        auto dst = output;
        for (; n; n -= 16, src += 32)
        {
                int16_t y[16] = {
                        src[0], src[2], src[4], src[6],
                        src[8], src[10], src[12], src[14],
                        src[16], src[18], src[20], src[22],
                        src[24], src[26], src[28], src[30],
                }, u[16] = {
                        src[1], src[1], src[5], src[5],
                        src[9], src[9], src[13], src[13],
                        src[17], src[17], src[21], src[21],
                        src[25], src[25], src[29], src[29],
                }, v[16] = {
                        src[3], src[3], src[7], src[7],
                        src[11], src[11], src[15], src[15],
                        src[19], src[19], src[23], src[23],
                        src[27], src[27], src[31], src[31],
                };

                uint8_t r[16], g[16], b[16];
                for (int i = 0; i < 16; i++)
                {
                        int32_t c = y[i] - 16;
                        int32_t d = u[i] - 128;
                        int32_t e = v[i] - 128;

                        int32_t t;
#define clamp(x)  ((t=(x)) > 255 ? 255 : t < 0 ? 0 : t)
                        r[i] = clamp((298 * c + 409 * e + 128) >> 8);
                        g[i] = clamp((298 * c - 100 * d - 208 * e + 128) >> 8);
                        b[i] = clamp((298 * c + 516 * d + 128) >> 8);
#undef clamp
                }

                uint8_t out[16 * 4] = {
                                r[0], g[0], b[0], 255, r[1], g[1], b[1], 255,
                                r[2], g[2], b[2], 255, r[3], g[3], b[3], 255,
                                r[4], g[4], b[4], 255, r[5], g[5], b[5], 255,
                                r[6], g[6], b[6], 255, r[7], g[7], b[7], 255,
                                r[8], g[8], b[8], 255, r[9], g[9], b[9], 255,
                                r[10], g[10], b[10], 255, r[11], g[11], b[11], 255,
                                r[12], g[12], b[12], 255, r[13], g[13], b[13], 255,
                                r[14], g[14], b[14], 255, r[15], g[15], b[15], 255,
                };
#ifdef _WIN32
                memcpy_s((void *)dst, sizeof out, out, sizeof out);
#else
                memcpy((void *)dst, out, sizeof out);
#endif
                dst += sizeof out;
        }
}

void CustomCalibration::RenderLines(std::vector<POINT_D> points, COLOR color, int
lineWidth)
{
        glEnable(GL_BLEND);
        glLineWidth((GLfloat)lineWidth);
        glColor4ub(color.R, color.G, color.B, color.A);

        for (int i = 0; i < points.size(); i += 2)
        {
                glBegin(GL_LINES);
                glVertex2d(points[i].x, points[i].y);
                glVertex2d(points[i+1].x, points[i+1].y);
                glEnd();
        }
        glDisable(GL_BLEND);
}

void CustomCalibration::RenderText(float x, float y, void *font, const char* text, COLOR
color)
{
        int w = glutGet(GLUT_WINDOW_WIDTH);
        int h = glutGet(GLUT_WINDOW_HEIGHT);
        float x0 = (x + 1) * w / 2;
        float y0 = h - (y + 1) * h / 2;

        if (x == -2) x0 = 20;

        if (x == -3 || x == -5)
        {
                int nStrLengthPixels = 0;
                for (char *p = (char*)text; *p; p++)
                {
                        if (*p == '\n') break;
                        nStrLengthPixels += glutBitmapWidth(font, *p);
                }

                if (x == -3)
                        x0 = w - nStrLengthPixels - 20;
                else
                        x0 = (w - nStrLengthPixels) / 2;
        }
        glPushAttrib(GL_ALL_ATTRIB_BITS);
        glMatrixMode(GL_PROJECTION);
        glPushMatrix();
        glLoadIdentity();
        glOrtho(0, glutGet(GLUT_WINDOW_WIDTH), glutGet(GLUT_WINDOW_HEIGHT), 0, -1, +1);
        glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
        glLoadIdentity();

        glColor4ub(color.R, color.G, color.B, color.A);
        glRasterPos2f(x0, y0);

        glutBitmapString(font, (const unsigned char*)text);

        // Restore GL state to what it was prior to this call
        glPopMatrix();
        glMatrixMode(GL_PROJECTION);
        glPopMatrix();
        glPopAttrib();
}
```

### 6.3.5 Rs400Device.cpp

```
/*
* INTEL CORPORATION PROPRIETARY INFORMATION
* This software is supplied under the terms of a license agreement
* or nondisclosure agreement with Intel Corporation and may not be
* copied or disclosed except in accordance with the terms of that
* agreement.
* Copyright(c) 2016-2017 Intel Corporation. All Rights Reserved.
*/

#ifndef _WIN32
#include <dirent.h>
#else
#include <process.h>
#endif

#include <iostream>
#include <string>
#include <cstring>
#include "Rs400Device.h"
#include "rs_advanced_mode.hpp"

using namespace std;
using namespace RsCamera;
using namespace rs2;
```

```
Rs400Device::Rs400Device()
{
    m_context = new context();

    m_captureStarted = false;
    m_stopProcessFrame = true;

        m_depthSensor = NULL;
        m_colorSensor = NULL;

    for (uint32_t i = 0; i < NELEMS(m_timestamp); i++)
    {
        m_timestamp[i] = 0;
        m_pData[i] = nullptr;
    }

    m_ts = 0x8000000000;

#ifdef _WIN32
    InitializeCriticalSection(&m_mutex);
#else
    if (0 != pthread_mutex_init(&m_mutex, NULL)) throw
std::runtime_error("pthread_mutex_init failed");
#endif
}

Rs400Device::~Rs400Device()
{
#ifndef _WIN32
    if (0 != pthread_mutex_destroy(&m_mutex)) throw
std::runtime_error("pthread_mutex_destroy failed");
#endif

    m_depthProfiles.clear();
        m_colorProfile.clear();
    delete m_context;
}


camera_info Rs400Device::InitializeCamera()
{
        camera_info info{};
        auto devices = m_context->query_devices();
    m_device = devices[0];

    if (m_device.is<rs400::advanced_mode>())
    {
                rs400::advanced_mode advanced = m_device.as<rs400::advanced_mode>();
        if (!advanced.is_enabled())
        {
            advanced.toggle_advanced_mode(true);
            //Remove context recreation after libRs fix
            delete m_context;
            m_context = new context();
            devices = m_context->query_devices();
                        m_device = devices[0];
        }
    }

        auto sensors = devices[0].query_sensors();
        m_depthSensor = sensors[0];

        info.name = m_device.get_info(rs2_camera_info::RS2_CAMERA_INFO_NAME);
        // filter out non RS400 camera
        if (info.name.find("RealSense") == string::npos || info.name.find("4") ==
string::npos) return info;

        info.pid = m_device.get_info(rs2_camera_info::RS2_CAMERA_INFO_PRODUCT_ID);
        info.serial = m_device.get_info(rs2_camera_info::RS2_CAMERA_INFO_SERIAL_NUMBER);
        info.fw_ver =
m_device.get_info(rs2_camera_info::RS2_CAMERA_INFO_FIRMWARE_VERSION);

        std::vector<uint8_t> RawBuffer =
        { 0x14, 0, 0xab, 0xcd, 0x10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0 };

        std::vector<uint8_t> rcvBuf;

        auto debug = m_device.as<debug_protocol>();;
        rcvBuf = debug.send_and_receive_raw_data(RawBuffer);

        if ((rcvBuf[SKU_COMPONENT] & 0x03) == 2)
                info.isWide = true;

        if (rcvBuf[RGB_MODE] & 0x01)
        {
                info.isRGB = true;
                if (sensors.size() > 1)
                {
                        m_colorSensor = sensors[1];
                }
        }

    return info;
}

bool Rs400Device::SetMediaMode(int width, int height, int frameRate, int colorWidth, int
colorHeight, bool enableColor)
{
    m_bColorEnabled = enableColor;

    m_depthProfiles.clear();
        m_colorProfile.clear();

    //Enable Y16 format for Left
    stream_profile infraredProfile;
    if (!GetProfile(infraredProfile, rs2_stream::RS2_STREAM_INFRARED, width, height,
frameRate, 1)) return false;

    m_depthProfiles.push_back(infraredProfile);

    //Enable Y16 format for Right
    stream_profile infraredProfile2;
    if (!GetProfile(infraredProfile2, rs2_stream::RS2_STREAM_INFRARED, width, height,
frameRate, 2)) return false;

    m_depthProfiles.push_back(infraredProfile2);

    if (enableColor)
    {
        stream_profile profile;
        if (!GetProfile(profile, rs2_stream::RS2_STREAM_COLOR, colorWidth, colorHeight,
frameRate, 0)) return false;

        m_colorProfile.push_back(profile);
    }

        m_lrImage[0] = std::unique_ptr<uint16_t[]>(new uint16_t[width*(height + 1)]);
        m_lrImage[1] = std::unique_ptr<uint16_t[]>(new uint16_t[width*(height + 1)]);

    return true;
}

void Rs400Device::StartCapture(std::function<void(const void *leftImage, const void
*rightImage,
    const void *depthImage, const uint64_t timeStamp)> callback)
{
    m_callback = callback;

    if (!m_callback)
        throw std::runtime_error("SetCallback() must be called before StartCapture()!");

    if (m_depthProfiles.size() == 0)
        throw std::runtime_error("SetMediaMode() must be called before StartCapture()!");

    if (m_captureStarted) return;

    m_depthSensor.open(m_depthProfiles);
    try
    {
        m_depthSensor.start([&](rs2::frame f) {
            auto profile = f.get_profile();
            auto stream_type = profile.stream_type();
            auto video = profile.as<video_stream_profile>();

            if (stream_type != rs2_stream::RS2_STREAM_INFRARED)
                return;

            RS_400_STREAM_TYPE rs400StreamType;
            rs400StreamType = (profile.stream_index() == 1) ? RS400_STREAM_INFRARED :
RS400_STREAM_INFRARED2;

            MUTEX_LOCK(&m_mutex);

            m_timestamp[(int)rs400StreamType] =

(uint64_t)f.get_frame_metadata(rs2_frame_metadata_value::RS2_FRAME_METADATA_TIME_OF_ARRIV
AL);

            m_ts = m_timestamp[(int)rs400StreamType];

            m_pData[(int)rs400StreamType] = (void *)f.get_data();

            int size = 2*video.width() * video.height();
            uint16_t *lr = m_lrImage[(int)rs400StreamType].get();
#ifdef _WIN32
            memcpy_s((void *)lr, size, m_pData[(int)rs400StreamType], size);
#else
            memcpy((void *)lr, m_pData[(int)rs400StreamType], size);
#endif
```

```
            if (m_timestamp[RS400_STREAM_INFRARED] != m_timestamp[RS400_STREAM_INFRARED2]
|| m_bColorEnabled)
            {
                MUTEX_UNLOCK(&m_mutex);
                return;
            }

            //process
            uint8_t *left = (uint8_t *)m_lrImage[0].get();
            uint8_t *right = (uint8_t *)m_lrImage[1].get();
            uint8_t *color = nullptr;

            m_callback(left, right, color, m_ts);

            MUTEX_UNLOCK(&m_mutex);
        });
    }
    catch (...)
    {
        m_depthSensor.close();
        throw;
    }

        if (m_colorProfile.size() > 0)
        {
                m_colorSensor.open(m_colorProfile);
                try
                {
                        m_colorSensor.start([&](rs2::frame f) {
                                auto profile = f.get_profile();
                                auto stream_type = profile.stream_type();
                                auto video = profile.as<video_stream_profile>();

                                if (stream_type != rs2_stream::RS2_STREAM_COLOR) return;

                                MUTEX_LOCK(&m_mutex);

                                m_timestamp[(int)RS400_STREAM_COLOR] =

        (uint64_t)f.get_frame_metadata(rs2_frame_metadata_value::RS2_FRAME_METADATA_TIME_
OF_ARRIVAL);

                                m_pData[(int)RS400_STREAM_COLOR] = (void *)f.get_data();

                                if (m_timestamp[(int)RS400_STREAM_COLOR] < m_ts)
                                {
                                        MUTEX_UNLOCK(&m_mutex);
                                        return;
                                }

                                //process
                                uint8_t *left = (uint8_t *)m_lrImage[0].get();
                                uint8_t *right = (uint8_t *)m_lrImage[1].get();
                                uint8_t* color = (uint8_t
*)m_pData[(int)RS400_STREAM_COLOR];

                                m_callback(left, right, color, m_ts);

                                m_ts = m_timestamp[(int)RS400_STREAM_COLOR];

                                MUTEX_UNLOCK(&m_mutex);
                        });
                }
                catch (...)
                {
                        m_colorSensor.close();
                        throw;
                }
        }


    m_captureStarted = true;
}

void Rs400Device::StopCapture()
{
    if (!m_captureStarted) return;

        if (m_colorProfile.size() > 0)
        {
                m_colorSensor.stop();
                m_colorSensor.close();
        }

    m_depthSensor.stop();
    m_depthSensor.close();

    m_captureStarted = false;
}


void Rs400Device::EnableAutoExposure(float value)
{
    m_depthSensor.set_option(rs2_option::RS2_OPTION_ENABLE_AUTO_EXPOSURE, value);
}

void Rs400Device::EnableEmitter(float value)
{
        if (m_depthSensor.supports(rs2_option::RS2_OPTION_EMITTER_ENABLED))
        {
                m_depthSensor.set_option(rs2_option::RS2_OPTION_EMITTER_ENABLED, value);
        }
}

void Rs400Device::SetAeControl(unsigned int point)
{
        rs400::advanced_mode advanced = m_device.as<rs400::advanced_mode>();
        STAEControl aeControl = { point };
        advanced.set_ae_control(aeControl);
}

bool Rs400Device::GetProfile(stream_profile& profile, rs2_stream stream, int width, int
height,
    int fps, int index)
{
        rs2::sensor sensor;
        rs2_format format;

        if (stream == rs2_stream::RS2_STREAM_INFRARED)
        {
                sensor = m_depthSensor;
                format = rs2_format::RS2_FORMAT_Y16;
```

```
            }
            else
            {
                    sensor = m_colorSensor;
                    format = rs2_format::RS2_FORMAT_YUYV;
            }

        vector<stream_profile> pfs = sensor.get_stream_profiles();

        for (int i = 0; i < (int)pfs.size(); i++)
        {
            auto video = pfs[i].as<video_stream_profile>();

            if ((pfs[i].format() == format)
                && (video.width() == width)
                && (video.height() == height)
                && (video.fps() == fps)
                && (video.stream_index() == index)
                )
            {
                profile = pfs[i];
                return true;
            }
        }

        return false;
    }
```