Omni Camera User Manual

Release 2.1

Occam Vision Group

Mar 20, 2019

CONTENTS:

CHAPTER

ONE

OVERVIEW

This is the manual for the Omni 60 and Omni Stereo cameras. It includes specification data for each camera, set up and usage instructions for Windows and Linux, and also covers more advanced topics like camera calibration, synchronization, and troubleshooting.

The Omni 60 camera is a 5-sensor 360-degree view camera that captures video at 60 frames per second (FPS) in raw format to a Windows or Linux computer. The Omni Stereo camera is a 10-sensor two-row 360 camera that emits 360-capture video like the Omni 60 as well as stereo from 5 vertical stereo pairs, enabling omnidirectional stereo capture.

The audience of the document is any user of the Omni cameras, including developers as well as non-developers. Development in C/C++ is required for most applications beyond simple recording which can be done through the Indigo Tools software.

There are three primary components included in the software: the Indigo Tools software (Windows only), the Indigo SDK (Windows and Linux), and the ROS module (Linux only). The Indigo SDK and ROS module have BSD-licensed source code provided. The Indigo Tools software is simple to install and use to test and visualize camera output, and the latter two packages are used to programmatically control the camera and are normally how users integrate the camera into their projects.

CHAPTER

TWO

TECHNICAL SPECIFICATIONS

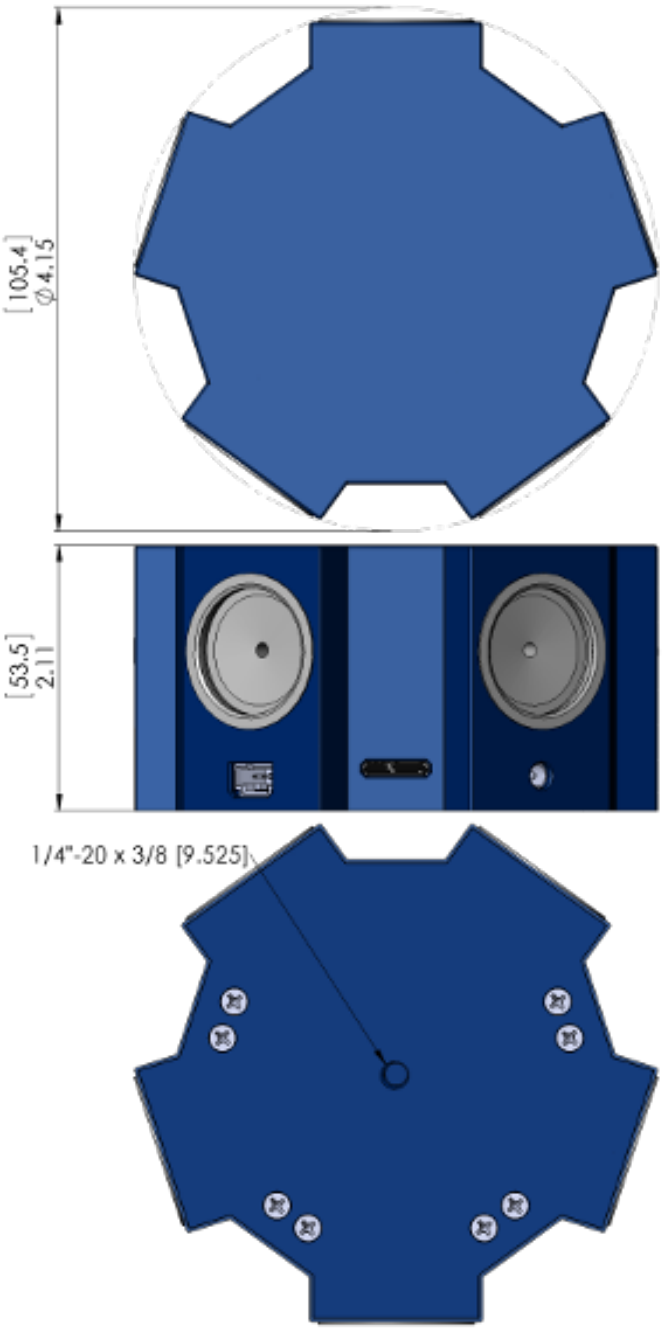
2.1 Omni 60

The Omni 60 is a 5-sensor omnidirectional camera that captures 360-degree video at 60 frames per second.

2.1.1 Specifications

MODEL	OMNI5U3MT9V022M	OMNI5U3MT9V022C
Resolution	1.8 MP (5x 752x480)	1.8 MP (5x 752x480)
Max Frame Rate	60 FPS	60 FPS
Horizontal FOV	360 degrees	360 degrees
Vertical FOV	58 degrees	58 degrees
Interface	USB3	USB3
Max Cable Length	15 feet (included)	15 feet (included)
Color	No	Yes
Sensor	5x Aptina MT9V022	5x Aptina MT9V022
Trigger Input	Yes	Yes
Trigger Output	Yes	Yes
Manual Exposure Control	Yes	Yes
Automatic Exposure Control	Yes	Yes
Manual Gain Control	Yes	Yes
Automatic Gain Control	Yes	Yes
Blending	CPU and GPU implementations (OpenGL)	
Stitching	Precalibrated; presets selectable during operation	
Lens	5x 2.8mm F2.0	5x 2.8mm F2.0
DC Input (optional)	5 VDC input	5 VDC input
Power Consumption	400mA	400mA
GPIO	1 input, 1 output, ground	
OS Support	Windows 7, 8, 10, Linux 3.x+	
Width	4.1”	4.1”
Height	4.1”	4.1”
Depth	2.1”	2.1”
Weight	13.3 oz (377 g)	13.3 oz (377 g)
Weight (without enclosure)	2.8 oz (79.3 g)	2.8 oz (79.3 g)

2.1.2 Mechanical and 3D Drawings



Please refer to the file *occam-omni60-dims.pdf* included with this package for a more detailed mechanical drawing of the Omni 60.

Please refer to the file *omni-60.stl* included with this package for the STL format CAD model of the camera.

Please refer to the file *omni-60.zip* included with this package for the STEP format CAD model of the camera.

2.2 Omni Stereo

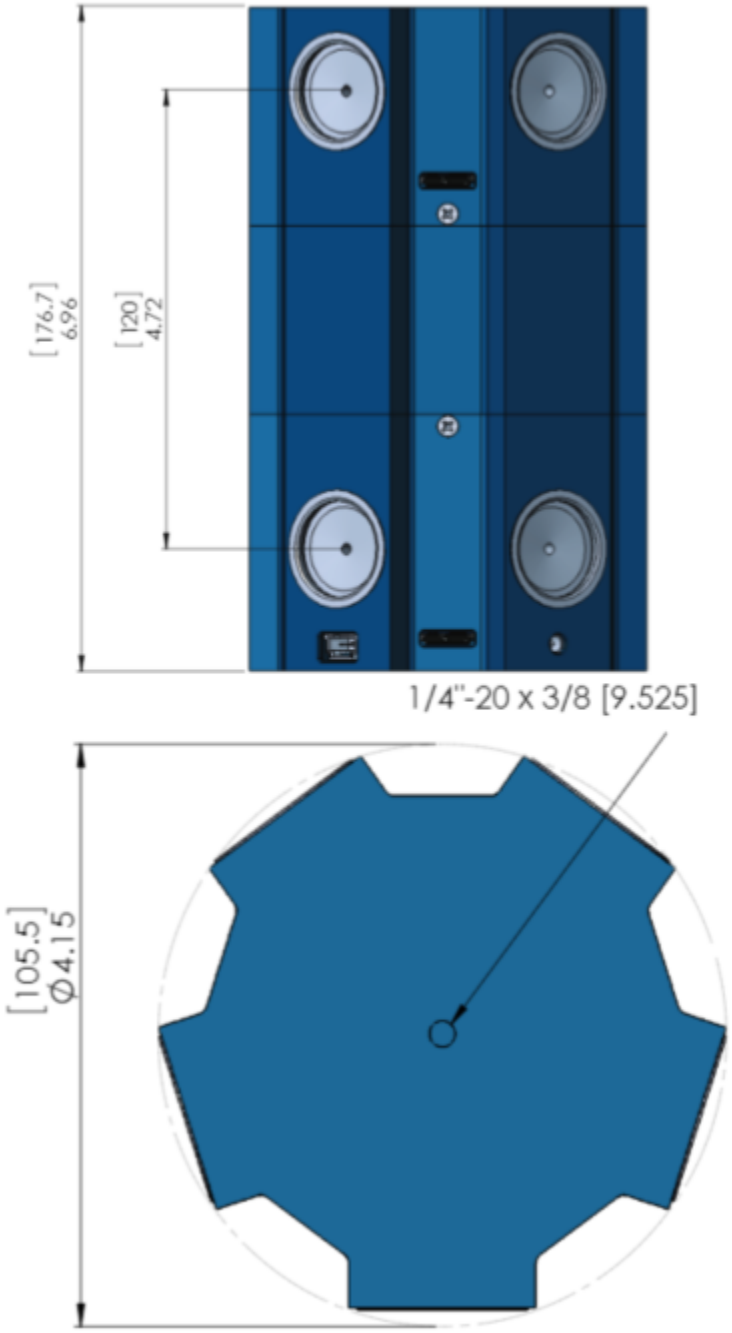
The Omni Stereo is a 10-sensor omnidirectional camera that captures 360-degree video at 60 frames per second.

Internally, the camera is made up of two Omni 60 cameras that are synchronized. Corresponding sensor pairs from the top and bottom Omni 60 provide 5 stereo pairs which enable 360-degree depth imaging.

2.2.1 Specifications

MODEL	OMNIS5U3MT9V022M	OMNIS5U3MT9V022C
Resolution	3.6 MP (10x 752x480)	3.6 MP (10x 752x480)
Max Frame Rate	60 FPS	60 FPS
Horizontal FOV	360 degrees	360 degrees
Vertical FOV	58 degrees	58 degrees
Baseline	4.72” (12cm)	4.72” (12cm)
Interface	USB3	USB3
Max Cable Length	15 feet (included)	15 feet (included)
Color	No	Yes
Sensor	10x Aptina MT9V022	10x Aptina MT9V022
Trigger Output	Yes	Yes
Manual Exposure Control	Yes	Yes
Automatic Exposure Control	Yes	Yes
Manual Gain Control	Yes	Yes
Automatic Gain Control	Yes	Yes
Stitching/Blending	CPU and GPU implementations (OpenGL)	
Lens	10x 2.8mm F2.0	10x 2.8mm F2.0
DC Input (optional)	5 VDC input	5 VDC input
Power Consumption	800mA	800mA
GPIO	1 input, 1 output, ground	
OS Support	Windows 7, 8, 10, Linux 3.x+	
Width	4.1”	4.1”
Height	6.96”	6.96”
Weight	39.8 oz (1128 g)	39.8 oz (1128 g)

2.2.2 Mechanical and 3D Drawings



Please refer to the file *occam-omnistereo-dims.pdf* included with this package for a more detailed mechanical drawing of the Omni Stereo.

Please refer to the file *omni-stereo.zip* included with this package for the STEP format CAD model of the camera.

2.3 Sensor

The sensor type is MT9V022, which supports monochrome and color capture at 60 FPS or lower, and has a resolution of 752 x 480.

For the Omni 60 which has five sensors, this translates to a total camera resolution of 1.8 megapixels. For the Omni Stereo which has ten sensors, this translates to a total camera resolution of 3.6 megapixels.

2.4 Field of View

The horizontal field of view of the individual lenses is about 80 degrees, and the vertical field of view of the of the individual lenses is about 58 degrees. The total camera horizontal field of view is thus 360 degree in aggregate, and 58 degrees vertical.

2.5 Baseline

The Omni Stereo internally is composed of two Omni 60 cameras stacked vertically. Each of the sensors of the top inner camera is associated with the bottom sensor directly below it, and forms a stereo pair. Thus the Omni Stereo has five stereo pairs.

Each stereo pair has a baseline of 12 cm.

Because the orientation of the stereo pairs is vertical, stereo matching proceeds on the transposed images. Therefore the matching happens on a pair of rectified 480 x 752 images.

2.6 Color

The color variants of the Omni 60 and Omni Stereo use bayer pattern to capture color. This is a repeating 2x2 pixel mask that sits over the sensor that allows different pixels to respond only to certain wavelengths. The standard bayer pattern 2x2 block has two green pixels in the diagonal, one blue pixel, and one red pixel.

From the bayer pattern a full RGB image is derived by interpolating the values for each color component from the neighboring pixels. Note that the RGB image is 3 times larger than the raw bayer image, and the raw bayer image is the same size of the monochrome image (both are 752 * 480 = 360,960 bytes).

2.7 Trigger Input and Output

The Omni 60 and Omni Stereo support trigger output, where one of the GPIO pins indicates the frame capture start time.

The Omni 60 also supports trigger input up to a capture rate of 30 FPS.

2.8 Exposure/Gain Control

The sensor supports automatic exposure control as well as automatic gain control. This is performed at the level of each individual sensor.

2.9 Stitching/Blending

The SDK supports cylinderical blending of the five sensors. This projects the images to a cylinder formed around the camera in its calibration coordinate space.

2.10 Operating Systems Supported

On Windows, Windows 7 and higher are supported.

On Linux, 3.x and higher are supported.

2.11 Power Input

The Omni 60 includes a 5 VDC barrel power connector that can be used to power the camera in cases where the USB connection is underpowered. The Omni Stereo includes two barrel connectors to power each half of the camera.

Normally it is not required to separately power the camera, and it is preferred to use a powered USB hub if more power is needed.

2.12 GPIO

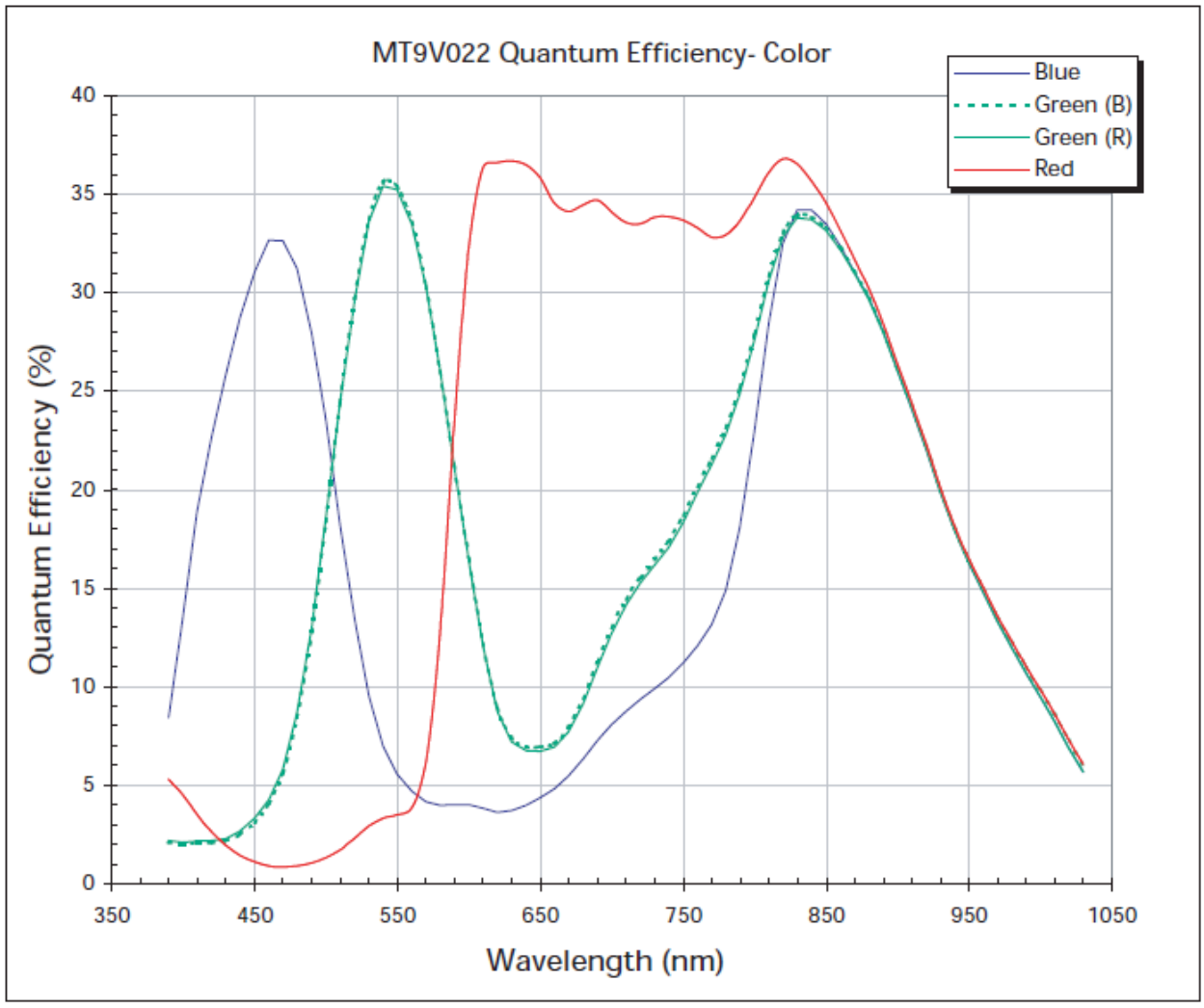
Three GPIO pins are provided on the Omni 60: input, output, and ground. These are non-isolated 3.3V TTL-level signals that provide trigger input, trigger output (indicating when frame capture started), and ground.

The Omni Stereo provides the same set of three pins on each half of the camera. Note that trigger input is not supported on the Omni Stereo because of the synchronization between the two halves of the camera.

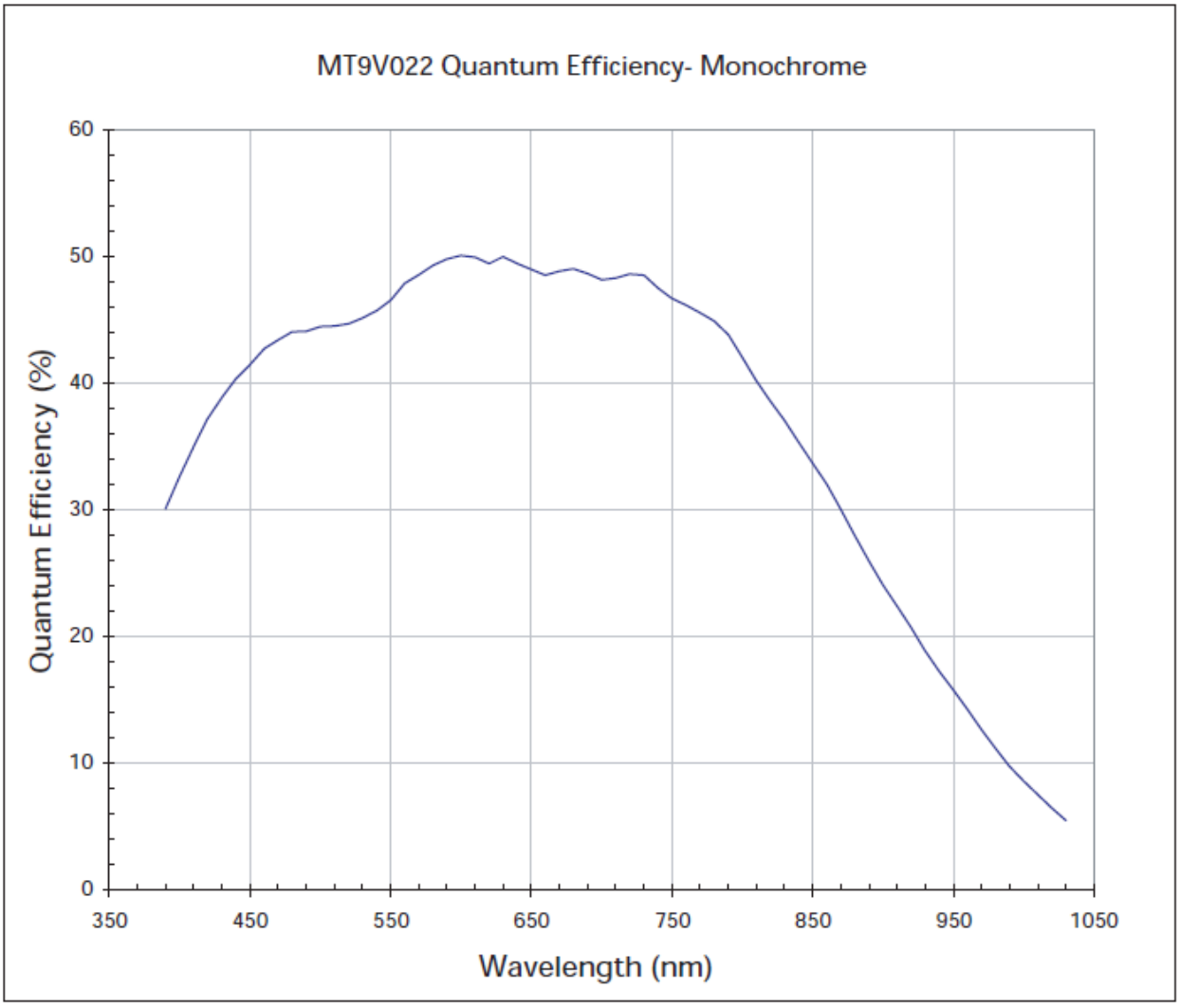
See the section regarding camera trigger functionality for more information about the input and output trigger.

2.13 Quantum Efficiency

For the color version of the cameras (omni5u3mt9v022c and omnis5u3mt9v022c):



For the monochrome version of the camera (omni5u3mt9v022m and omnis5u3mt9v022m):


MT9V022 Quantum Efficiency- Monochrome
Quantum Efficiency (%)
60
50
40
30
20
10
0
350
450
550
650
750
850
950
1050
Wavelength (nm)

CHAPTER

THREE

SYSTEM REQUIREMENTS

3.1 Windows

Windows 7 or higher is required. For building the SDK, Visual C++ 2013 or higher is required.

3.2 Linux

The SDK supports Ubuntu 12.04/14.04 and higher, and the specific commands and packages listed below correspond to these two versions. Generally speaking however, the SDK will run on any Linux distribution that has an up to date Linux kernel (anything newer than 3.4 should work), but you may have to make some adjustments to the code depending on the version of gcc or other differences between Ubuntu and your environment.

Here are two known-working kernels:

```
14.04.2: Linux occam-14-04 3.13.0-49-generic #83-Ubuntu SMP Fri Apr 10 20:11:33 UTC␣
↪2015 x86_64 x86_64 x86_64 GNU/Linux

12.04.5: Linux occam-12-04 3.13.0-49-generic #81~precise1-Ubuntu SMP Wed Mar 25␣
↪16:32:15 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
```

Use lspci -vv to see what USB3 controllers you have, and their kernel drivers. The built-in xhci_hcd driver has been stable since around kernel 3.4, and most USB3 controllers will work reliably with it and the Occam cameras.

Note that for Omni Stereo and Array multi-camera configurations, synchronization between cameras is achieved using USB3 ITP packets. This requires that all the cameras are plugged into the same USB3 controller.

Also note that total bandwidth available inbound from the USB3 controller must exceed the bandwidth required by the camera configuration. For Omni Stereo configurations, this requires that at least 2 PCIe lanes are available, since total camera bandwidth is about 210 MB/s where 1 PCIe lane only provides around 150 MB/s. You can use lspci to determine the number of lanes available to your USB3 controller. Most motherboard build-in controllers have 4 or more lanes, but it all depends on your particular hardware.

Here is example lspci output that shows a number of known-working controllers.

```
03:00.0 USB controller: Renesas Technology Corp. uPD720201 USB 3.0 Host
Controller (rev 03) (prog-if 30 [XHCI])
Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop-
ParErr- Stepping- SERR+ FastB2B- DisINTx+
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort-
SERR- Latency: 0, Cache Line Size: 64 bytes
Interrupt: pin A routed to IRQ 16
Region 0: Memory at fe900000 (64-bit, non-prefetchable)
[size=8K]
```

```
Capabilities:
Kernel driver in use: xhci_hcd
```

```
00:10.0 USB controller: Advanced Micro Devices, Inc. [AMD] FCH USB XHCI
Controller (rev 09) (prog-if 30 [XHCI])
Subsystem: Hewlett-Packard Company Device 2b17
Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop-
ParErr- Stepping- SERR+ FastB2B- DisINTx+
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort-
SERR- Latency: 0, Cache Line Size: 64 bytes
Interrupt: pin A routed to IRQ 18
Region 0: Memory at feb6a000 (64-bit, non-prefetchable)
[size=8K]
Capabilities:
Kernel driver in use: xhci_hcd
```

```
00:10.1 USB controller: Advanced Micro Devices, Inc. [AMD] FCH USB XHCI
Controller (rev 09) (prog-if 30 [XHCI])
Subsystem: Hewlett-Packard Company Device 2b17
Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop-
ParErr- Stepping- SERR+ FastB2B- DisINTx+
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort-
SERR- Latency: 0, Cache Line Size: 64 bytes
Interrupt: pin B routed to IRQ 17
Region 0: Memory at feb68000 (64-bit, non-prefetchable)
[size=8K]
Capabilities:
Kernel driver in use: xhci_hcd
```

3.3 CPU requirements

The output of both the Omni 60 and Omni Stereo is raw image data. For Omni 60, the camera emits a single frame F times per second that contains 5 sub-frames. The sub-frames are the 752 x 480 images of the 5 sensors in the camera, and are hardware synchronized. F can be configured to 15, 30, 45, or 60.

On Omni Stereo, there are two connections to the host PC: one from the top Omni 60 camera, and one from the bottom Omni 60 camera. The two halves of the camera are synchronized in hardware using USB3 ITP, and are required to be connected to the same USB3 controller on the host computer. Therefore the output of the Omni Stereo is 2 frame buffers, one from each half of the camera, each containing 5 sub-frames for a total of 10 sub-frames.

The sub-frames for individual sensors are always in raw format. For monochrome cameras, this means a normal black-and-white intensity image. For color cameras, it means a bayer pattern image. Bayer pattern needs to be converted to RGB and this process happens on the host PC in the SDK.

All further processing, including white balance, gamma correction, stitching, stereo processing, and so on, is done entirely on the host system. For full rate output of Omni 60, it is recommended to use a fast i7 system. For Omni Stereo, a fast i7 system will generate about 10 - 15 frames per second of final output if all outputs are enabled.

CHAPTER

FOUR

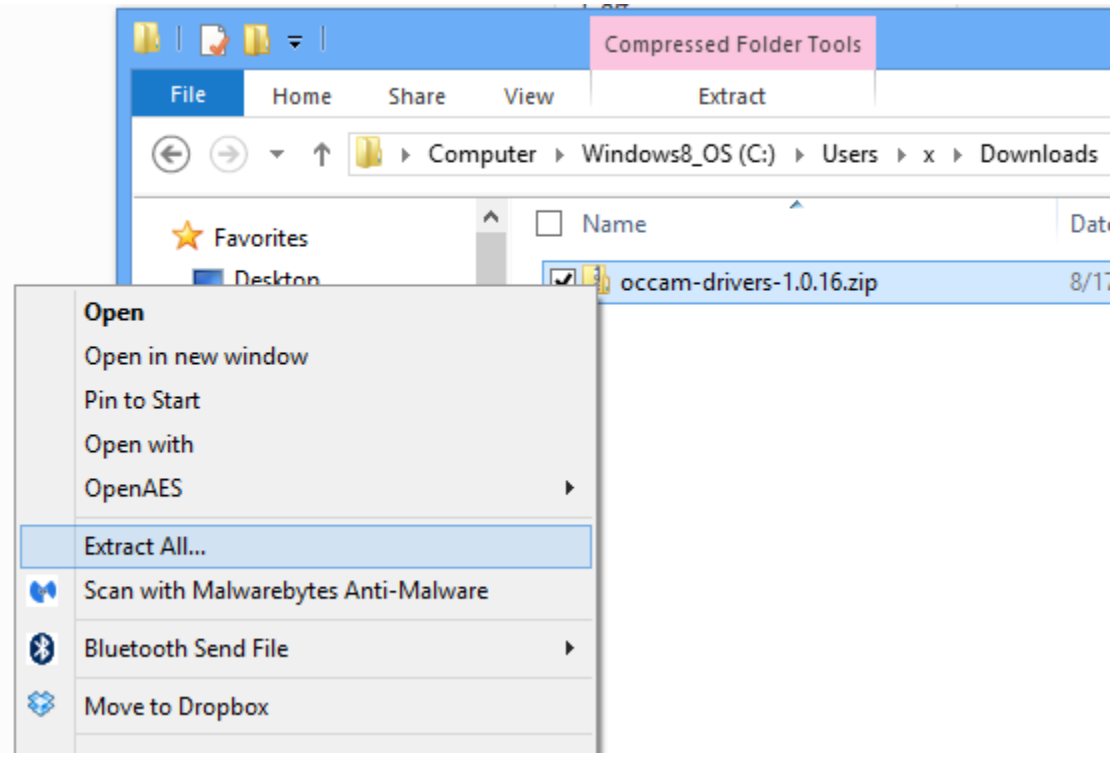
GETTING STARTED: INDIGO TOOLS

Indigo Tools is a UI software that provides easy access to the camera as well as recording functionality, and is a fast way to get up and running with an Omni camera. Indigo Tools is only available for Windows. For instructions about getting started on Linux, please see the next chapter.

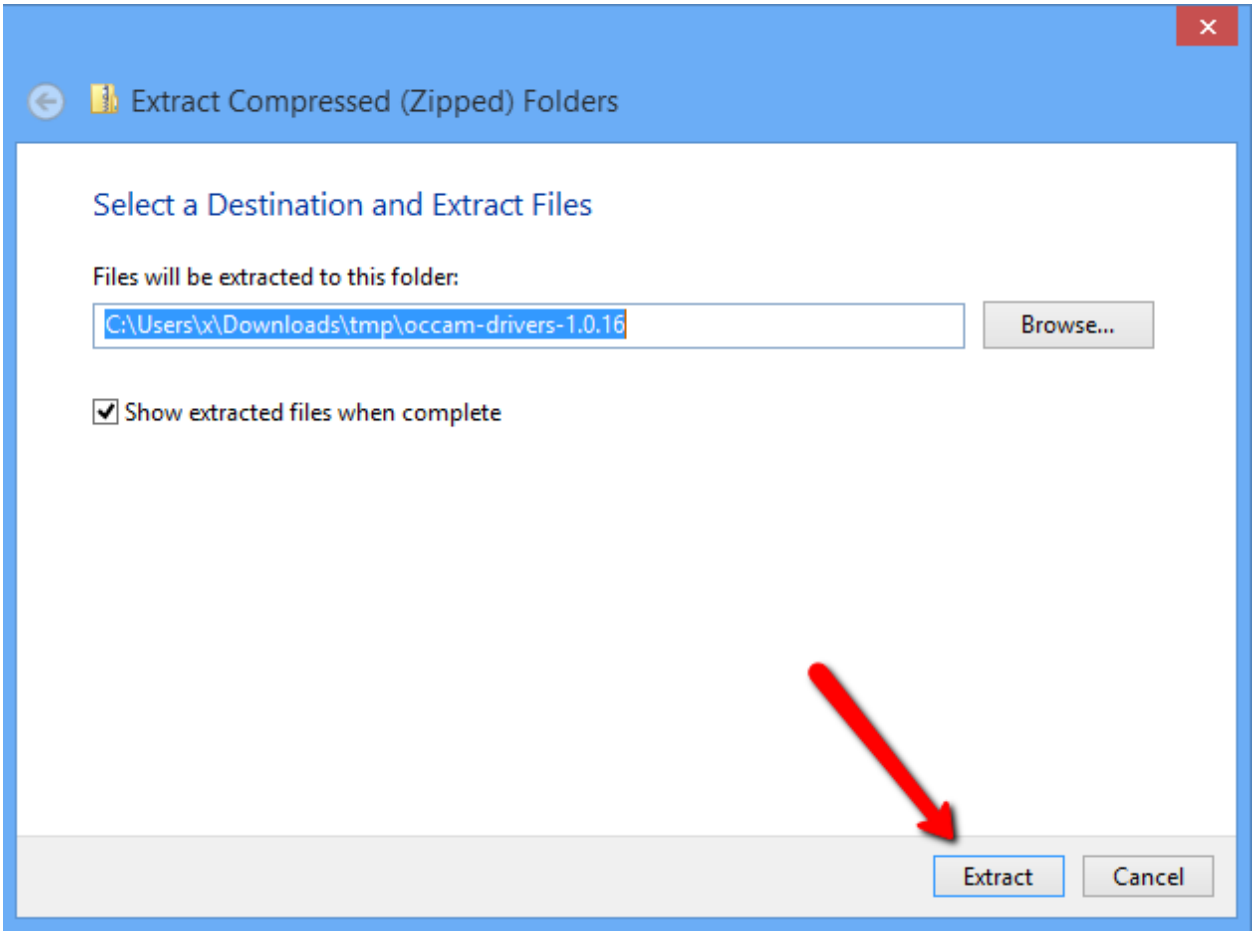
This section walks through how to install the driver for the Omni 60 and Omni Stereo cameras, and how to install the Indigo Tools software on your Windows PC. The instructions below are for Windows 8 and higher, but they're essentially identical for Windows 7. Some of the screens will be slightly different.

First, go download both the Windows Drivers and Indigo Tools and SDK from the Windows Drivers and Tools Downloads page on the occamvisiongroup.com web site, or retrieve them from the offline media that came with the camera.

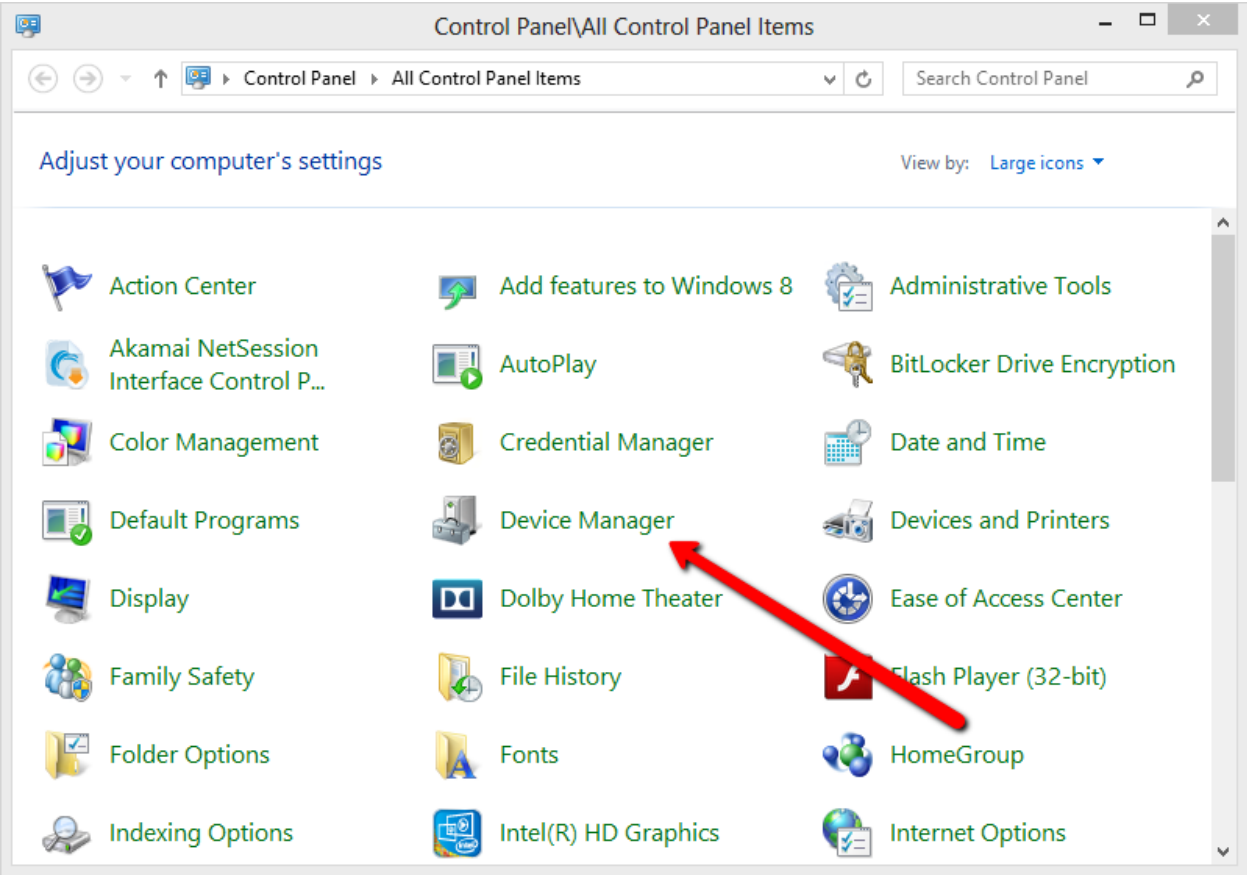
The driver package is a zip file that you will need to uncompress. This can be done using the built-in Windows zip tools, by right clicking and selecting "Extract All" from the menu.



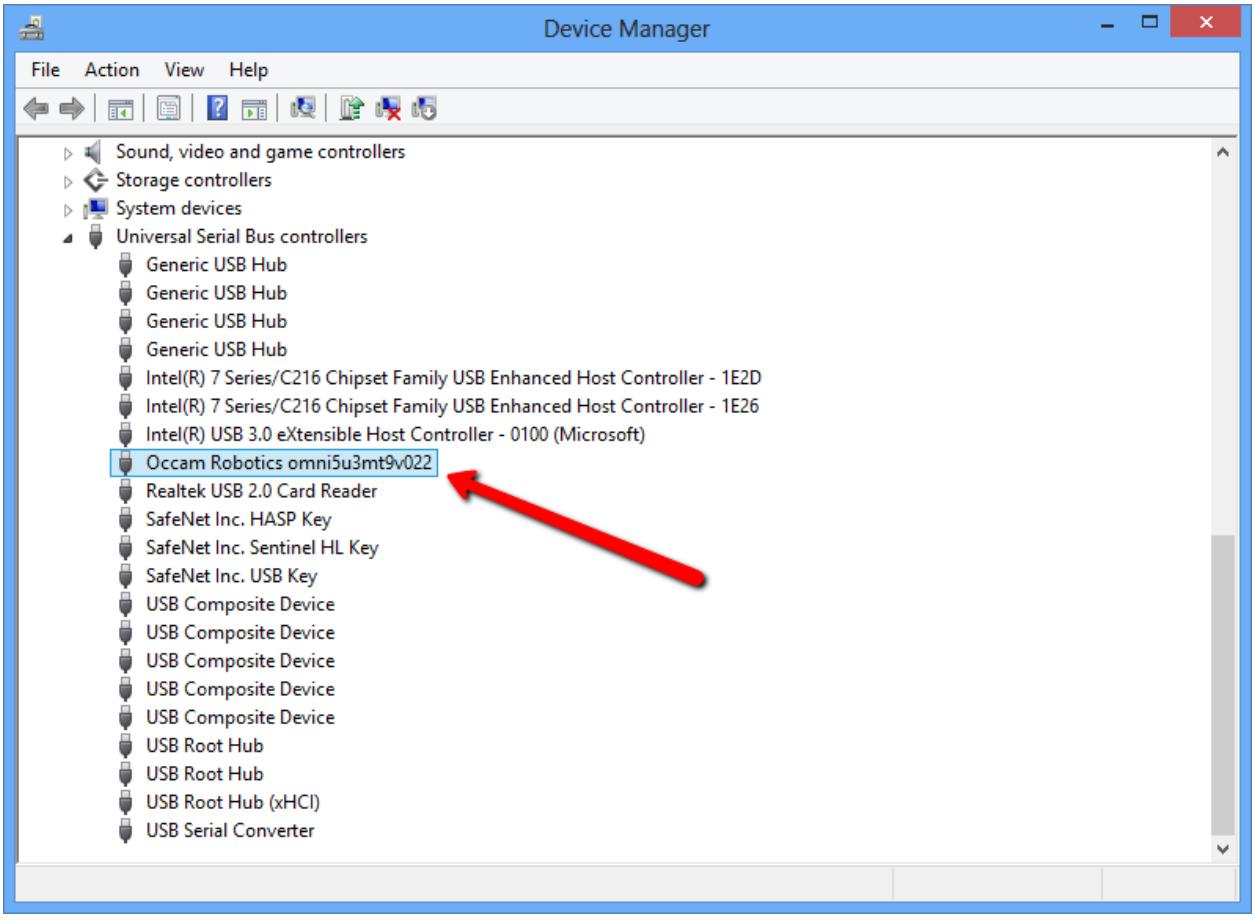
Click "Extract" to put the files into a new subdirectory from where you downloaded them.



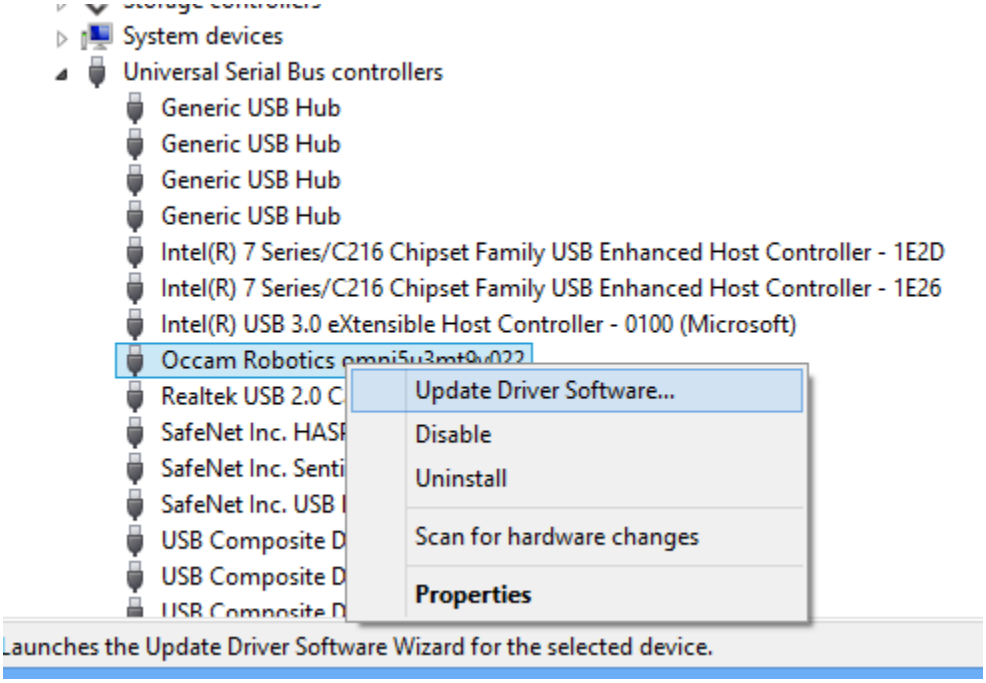
Now plug in the camera into a USB3 port on your computer and open the Device Manager.



You should see a device representing the Omni 60. It should appear as “Occam Robotics omni5u3mt9v022” or something similar.

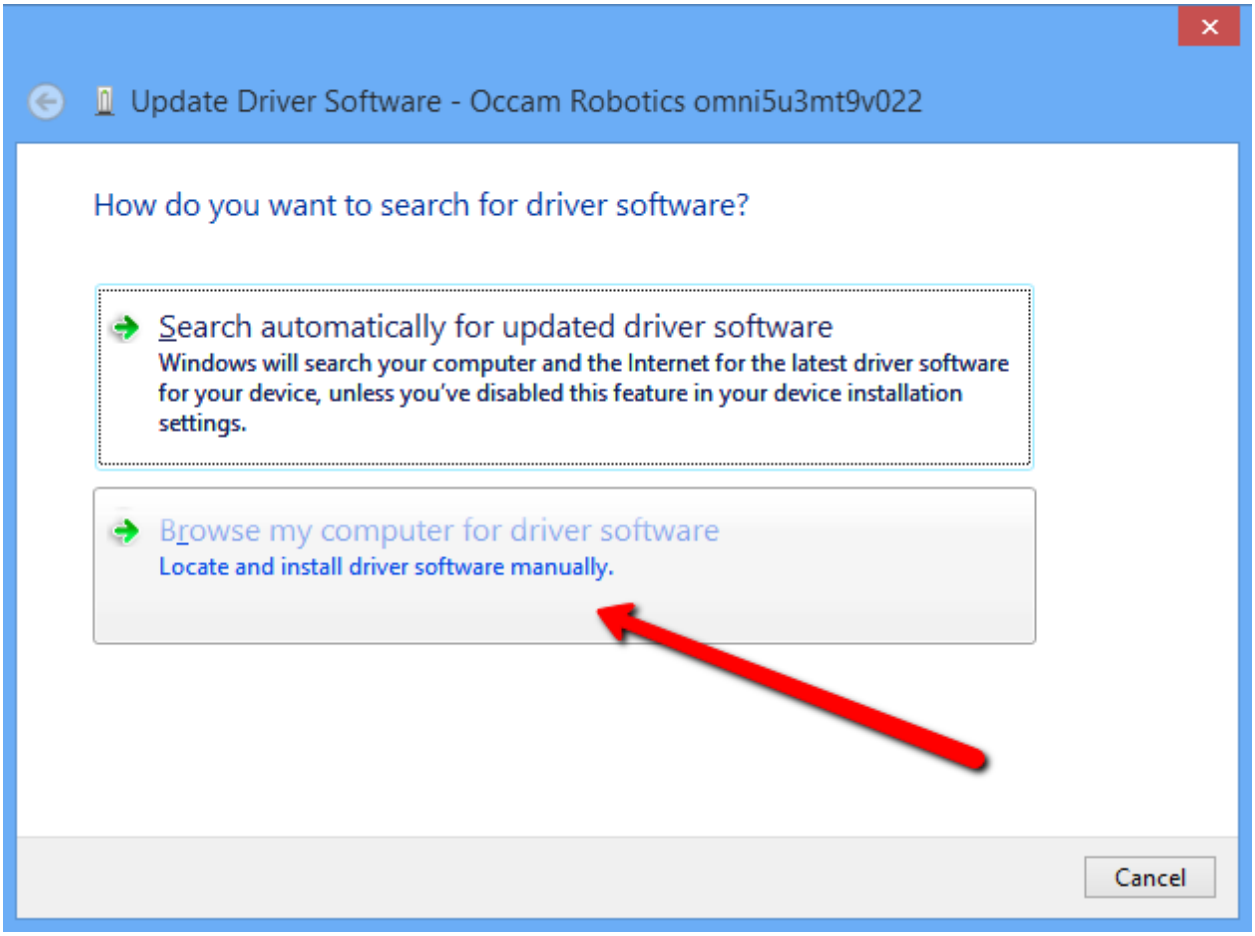


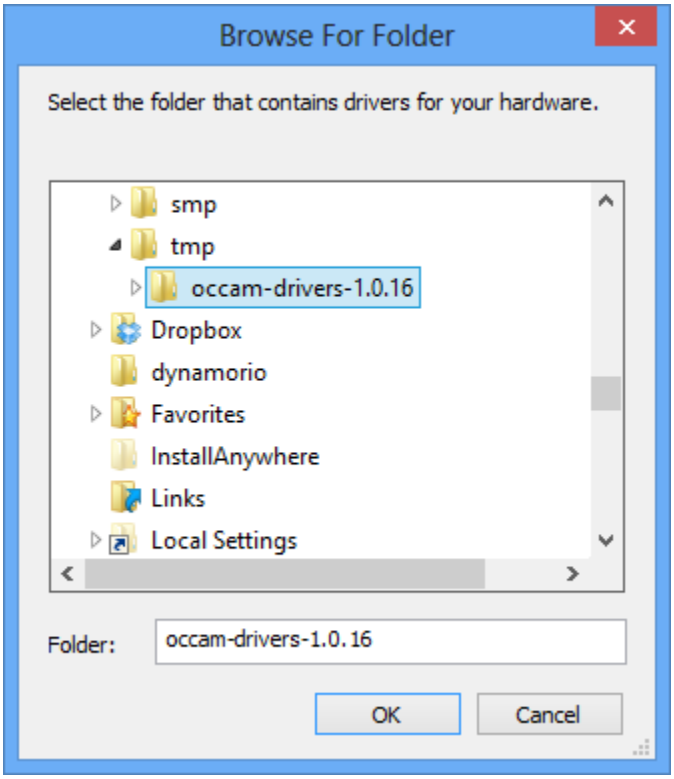
Right-click on the Omni camera device and select “Update Driver Software…”.



That will show a wizard asking you where the system can find the driver files. Go ahead and choose “Browse my

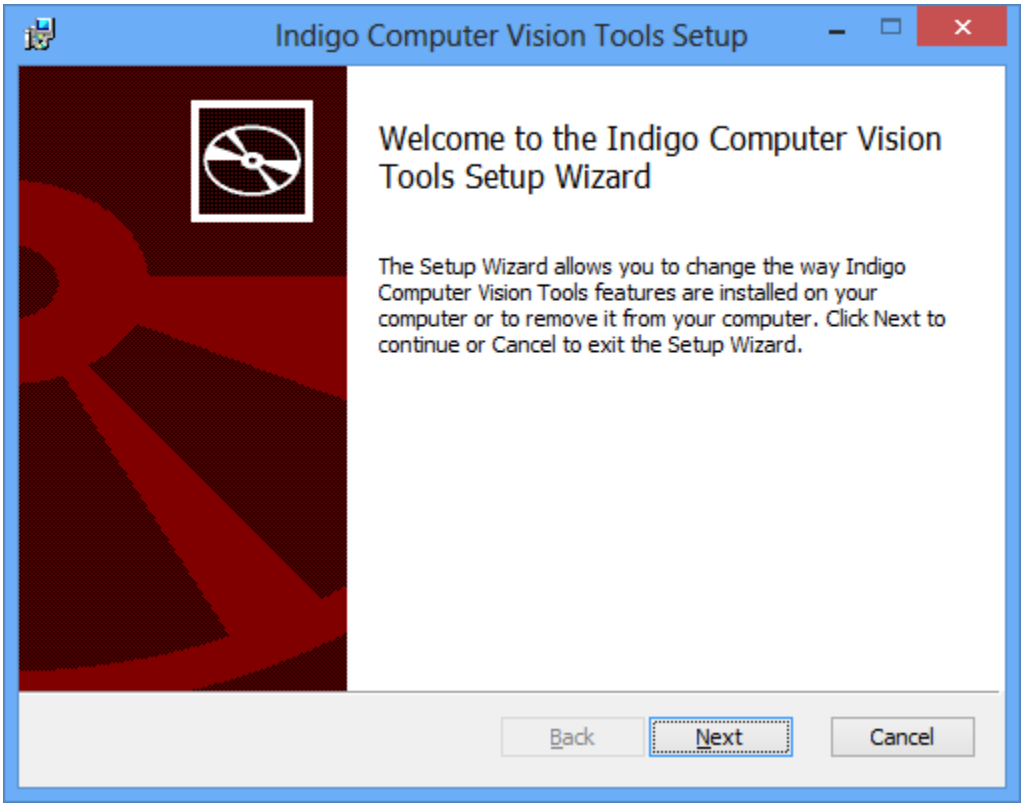
computer for driver software" and then select the directory where you unzipped the files.



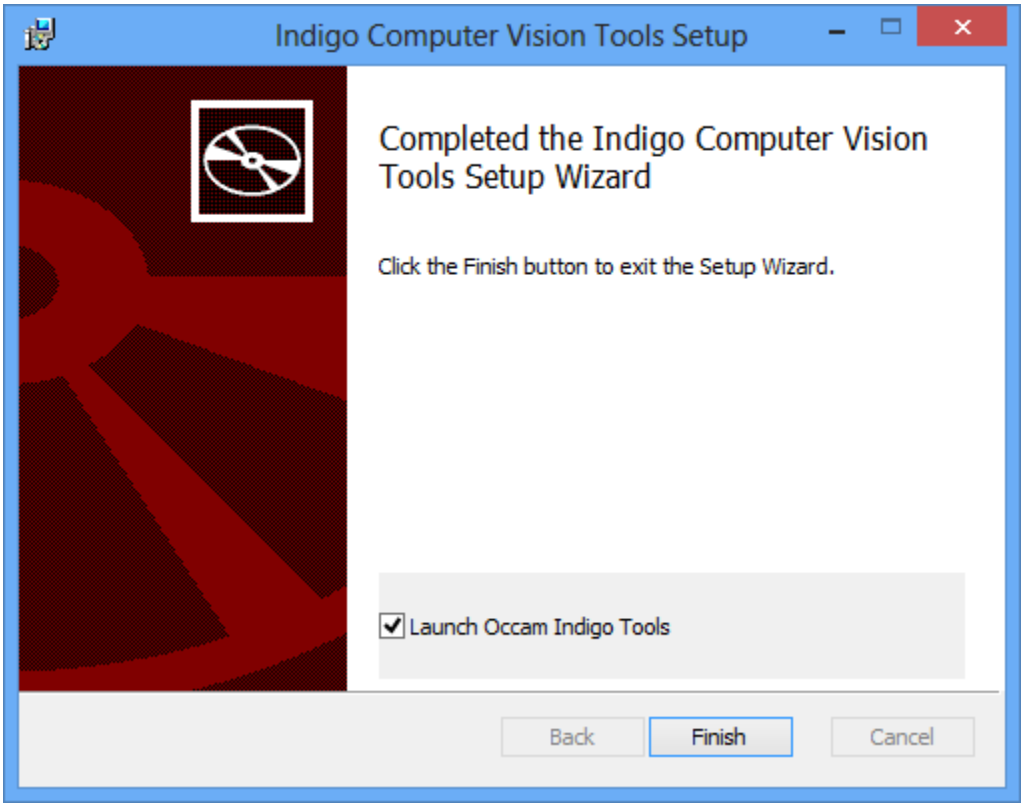


Click next several times and the driver should now be installed.

Now double-click on the installer for the Indigo Tools and SDK. That will launch the installation wizard:



Click next through all the screens to leave the installation directory and other options at their default. On the last screen of the wizard it will ask you whether you want to open the Indigo Tools software now. Leave that checked and click to close the wizard.



The tools should open and show a screen with the camera video output.

CHAPTER

FIVE

GETTING STARTED: INDIGO SDK

5.1 Building the SDK

This section walks through how to configure the Indigo SDK and get an Omni camera up and running.

Building the SDK (Software Development Kit) is relatively simple: simply run cmake to generate makefiles and then run make (or msbuild/Visual C++ build on Windows). The build products, including the indigo shared library as well as all the example programs will appear in the bin subdirectory.

You can optionally enable OpenCV integration (which will cause building of the relevant examples showing OpenCV integration), as well OpenGL integration.

```
cmake -DUSE_OPENCV=1 -DUSE_OPENGL=0 -DOpenCV_DIR=`pwd`/../opencv/build
```

To build the generated project in Linux, run:

```
make -j 10 -k
```

To build the generated project in Windows, it depends on what compiler suite you have configured cmake to use. In the typical case, you just need to open the project in Visual Studio / Visual C++ and click build from within the IDE (or run the msbuild tool to do the same from the command line).

5.2 Updating udev to allow non-root access to devices

On Linux, USB devices appear as device files that are only accessible by root on the system. You need to alter udev to have it change permissions when a device is plugged in. Otherwise the actual device is only accessible by root, and when you try to run an SDK program it will appear as though the camera is not plugged in, even though you can see the device when you run lsusb -vv.

You can also do this manually if needed.

A separate article discusses this briefly, here.

Basically you must create a file:

```
/etc/udev/rules.d/occam.rules
```

And add this line to it:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="285e", ATTRS{idProduct}=="3efd", MODE="0666"
```

Then restart udev:

```
sudo service udev restart
```

Then unplug and re-plug the camera, and it should appear with permissions 666, allowing any user on the system to talk to the camera with libusb.

5.3 Reading and displaying output from the camera

The easiest way to read data from the camera and verify that things are working as expected is to run the read_images_opencv example from the SDK bin directory.

```
bin/read_images_opencv
```

This will display a highgui window showing camera output. Note that the SDK interface allows for the driver to provide varied outputs. You can cycle through them with the 1 and 2 keys in the example. Any host-side computation is only performed if is required by the requested data. I.e., if you request disparity image for the first stereo pair only, then stereo matching will not be performed for the other stereo pairs, etc.

If you have access to a Windows machine, you are also encouraged to use the Windows Indigo Tools UI application to display output from the camera. The application has all the possible device configuration options mapped to UI, so you can easily get an idea of what control is possible.

5.4 libusb issues on Ubuntu 12.04

On a fully dist-upgrade'ed Ubuntu 12.04, the version of libusb included is too old (1.0.9) and has problems reading USB3 streams at full speed. This problem will manifest itself as the frame rate of the device not exceeding USB2 data rates (which for Omni Stereo is about 12 fps, instead of the full 60 fps).

You can use the indigosdk-x.y.z/bin/read_images_fps example to determine the frame rate you are able to read from the device. Also of possible interest, you can uncomment the

```
#define DEBUG_DATA_RATES
```

at the top of src/omni_libusb.cc. This will cause the SDK to emit a line to stderr once per second indicating the fps and MB/s read raw from the USB pipes. If everything is working properly with your hardware, you will see about 104 MB/s for Omni 60, and about 210 MB/s for Omni Stereo.

To update libusb, you will need to manually build it, for example with something like:

```
tar xjvf ~/Downloads/libusb-1.0.19.tar.bz2
cd libusb-1.0.19
sudo apt-get install libudev-dev
./configure --prefix=`pwd`/inst
make
make install
export PKG_CONFIG_PATH=`pwd`/inst/lib/pkgconfig

cd indigosdk-2.0.8
rm -f CMakeCache.txt && cmake -DOpenCV_DIR=~/opencv/build -DUSE_OPENCV=1 -DUSE_
↪OPENGL=0 .
make
bin/read_images_fps
```

Verify that 1.0.19 libusb is used by using make VERBOSE=1 when building the SDK. E.g., the -L path for libusb should point to the manually built one.

CHAPTER

SIX

COMPATIBILITY

6.1 Minimal Dependencies

On Linux, the minimal dependencies required to build the SDK are cmake, gcc, and libusb.

```
sudo apt-get install build-essential git cmake libusb-1.0-0-dev
```

On Windows, the minimal dependencies are either msys or Visual Studio. These can be downloaded from their respective project pages and have their own installation packages.

The Indigo Tools software (Windows only) has no external dependencies.

6.2 OpenGL dependency

OpenGL is an optional dependency that allows some acceleration of stitching and rendering operations.

libxxf86vm-dev and other GL libraries are required for GL support, but it depends on your graphics card. Use mesa libraries for generic, though that likely won't be faster than the default non-OpenGL implementations provided in the SDK.

6.3 OpenCV dependency

OpenCV is an optional dependency that is used by some of the examples in the SDK. The core of the SDK itself does not require OpenCV, but can be easily integrated with it (as demonstrated in several of the examples) and the camera calibration data provided by the cameras is in the same format as used in OpenCV.

On Linux and Windows, there are binary packages that can be downloaded for all versions of OpenCV.

To manually build and install OpenCV, a process similar to the following can be followed.

```
cd
git clone https://github.com/Itseez/opencv.git
```

To get graphics working from the read_images_opencv example:

```
sudo apt-get install libgtk2.0-dev pkg-config
```

Then build opencv, for example with:

```
cd opencv
mkdir build
cd build
cmake ..
make -j 10 -k
cd ../..
```

CHAPTER

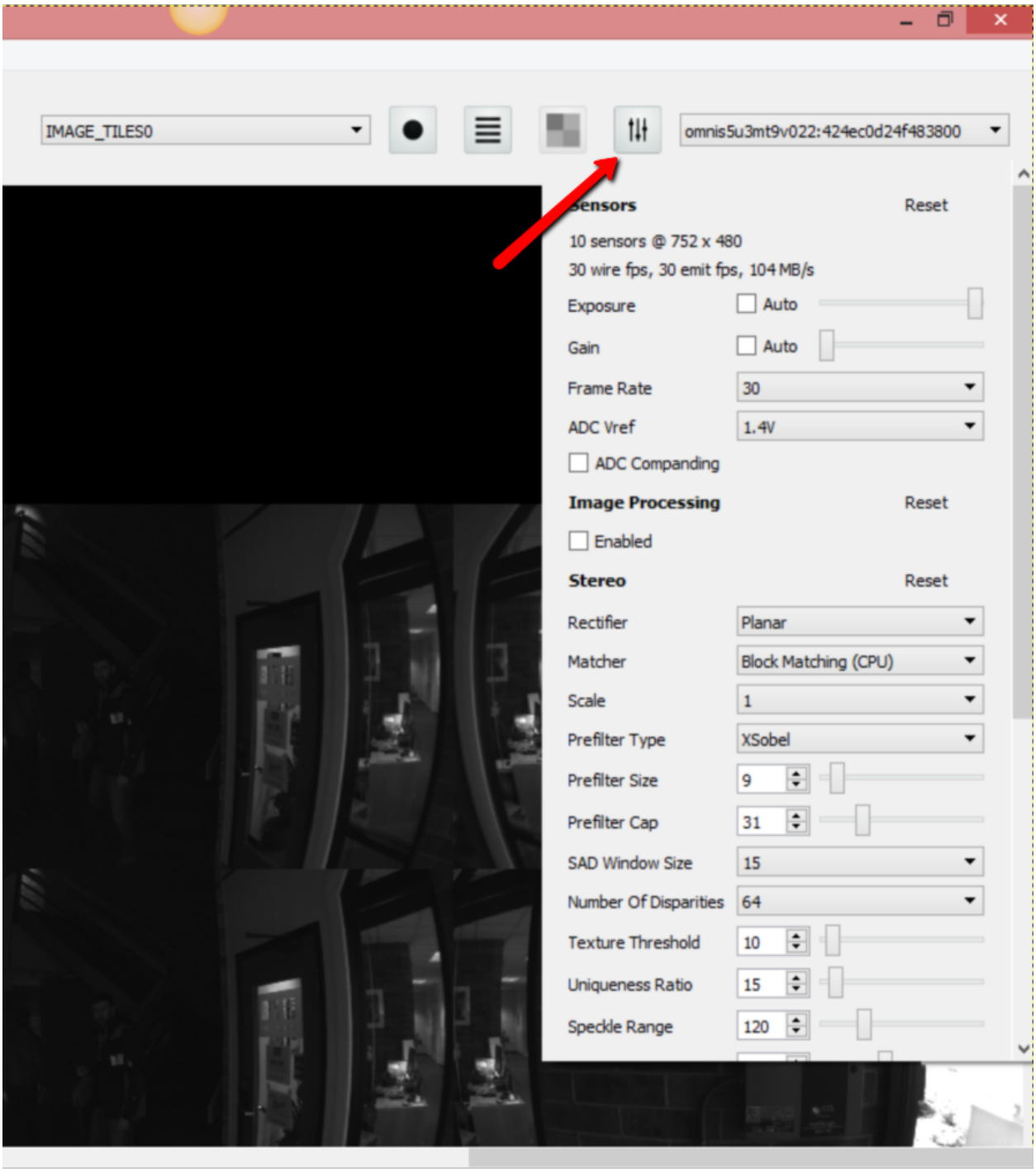
SEVEN

CAMERA CONTROL

7.1 Camera Control

When you plug in the Omni 60 / Omni Stereo, the driver will either use default settings (hard-coded in the driver) or load the settings from the camera's non-volatile storage, if available. To change any of the settings, click the control icon

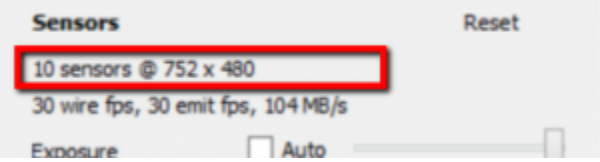
in the Indigo Tools toolbar:



That will open the full set of configurable settings available in the Omni Stereo driver. The settings you can control from this menu are the same set you can configure from the SDK interface with the occamSetDeviceParam*() function calls.

The full set of settings available depends on the driver as well as the set of plug-in modules loaded into the SDK. Modules that are attached to the device will expose settings of their own, and those will be accessible both from the Indigo Tools user interface as well as the SDK code interface. What follows is a walk-through of the settings available for the Omni Stereo device:

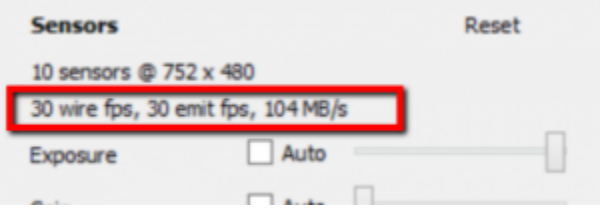
7.1.1 Sensor statistics



This displays the number of the sensors in the device as well as their resolutions. For fixed rigs such as the Omni series, the number of sensors is 5 or 10 depending on the model.

You can read these values using `occamGetDeviceValuei(device,OCCAM_SENSOR_COUNT, &value)`, `occamGetDeviceValuei(device,OCCAM_SENSOR_WIDTH,&value)`, and `occamGetDeviceValuei(device,OCCAM_SENSOR_HEIGHT,&value)`.

7.1.2 Driver statistics



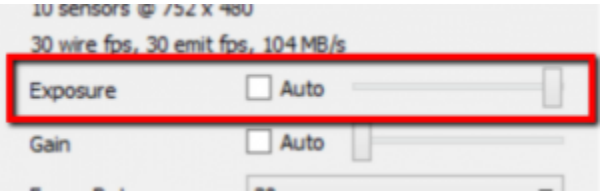
This indicates the total amount of data being read from the camera to the host, per second. This value will scale with the number of sensors in the camera, the resolution of the sensors, and the frame rate configured for them. If the camera is configured to run only in USB2 mode, or is bottlenecked in hardware for another reason, the value will be lower than it should be. Ideally the value is simply the sum of width*height*FPS for each sensor connected (plus a slight bit of negligible overhead).

Wire FPS refers to the frame rate received off the wire from USB, before any host-side image processing that happens (such as color conversion, image processing, image rectification or undistortion, stereo computations, blending, etc).

Emit FPS refers to the frame rate emited from the driver, after any host-side image processing. This value depends on what image processing is requested, in addition to the wire FPS. For example, if you request only `OCCAM_RAW_IMAGE0`, then no image processing will be performed at all, and only the raw image from sensor 0 will be returned. In this case the emit FPS should match the wire FPS.

You can read these values with `occamGetDeviceValuei(device,OCCAM_WIRE_FPS, &value)`, `occamGetDeviceValuei(device,OCCAM_WIRE_BPS,&value)`, and `occamGetDeviceValuei(device,OCCAM_EMIT_FPS,&value)`.

7.1.3 Exposure

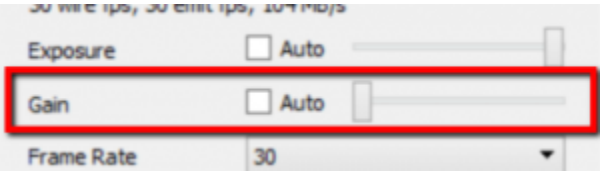


Use `occamSetDeviceValuei(device,OCCAM_EXPOSURE,VALUE)` to change the exposure. The units are in terms of sensor lines. You can use `occamSetDeviceValuei(device,`

`OCCAM_EXPOSURE_MICROSECONDS,VALUE)` to set the exposure in terms of microseconds. The driver will perform the sensor-specific conversion for you.

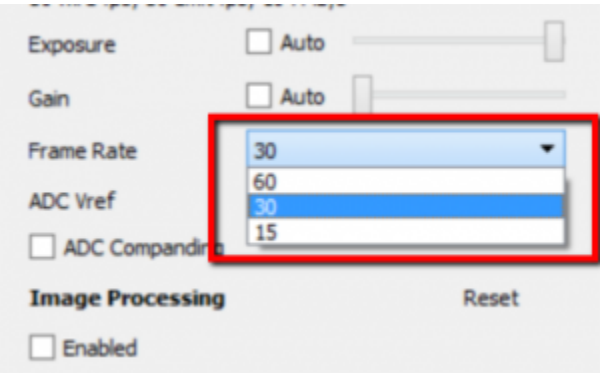
Use `occamSetDeviceValuei(device,OCCAM_AUTO_EXPOSURE,VALUE)` with `VALUE = 1` or `0` to enable or disable auto exposure.

7.1.4 Gain



The gain value control analog gain on the sensor, and the units depend on the specific sensor. You can set this value using `setOccamDeviceValuei(device,OCCAM_GAIN,VALUE)` to set the (analog) gain. The min and max can be queried by enumerating the parameters of the device. Use `setOccamDeviceValuei(device, OCCAM_AUTO_GAIN,VALUE)` for `VALUE = 1` or `0` to enable or disable automatic gain control.

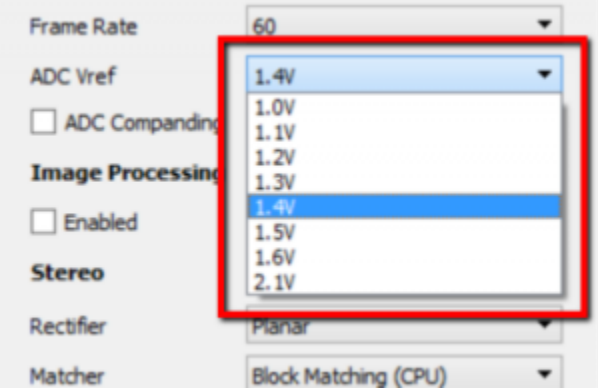
7.1.5 Frame Rate



The value controls the frame rate at which all the sensors will capture. The set of possible frame rates and video modes depends on the specific sensor and driver.

Use `occamSetDeviceValuei(device,OCCAM_TARGET_FPS,VALUE)` to set the target frame rate of the sensors. The permissible values depend on the sensor and can be queried with `occamGetDeviceValueiv`.

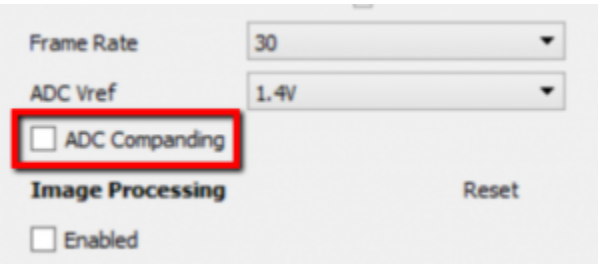
7.1.6 ADV Voltage Reference



This value controls the ADC reference voltage used when digitizing pixel values. The effect of this setting is similar to controlling analog gain.

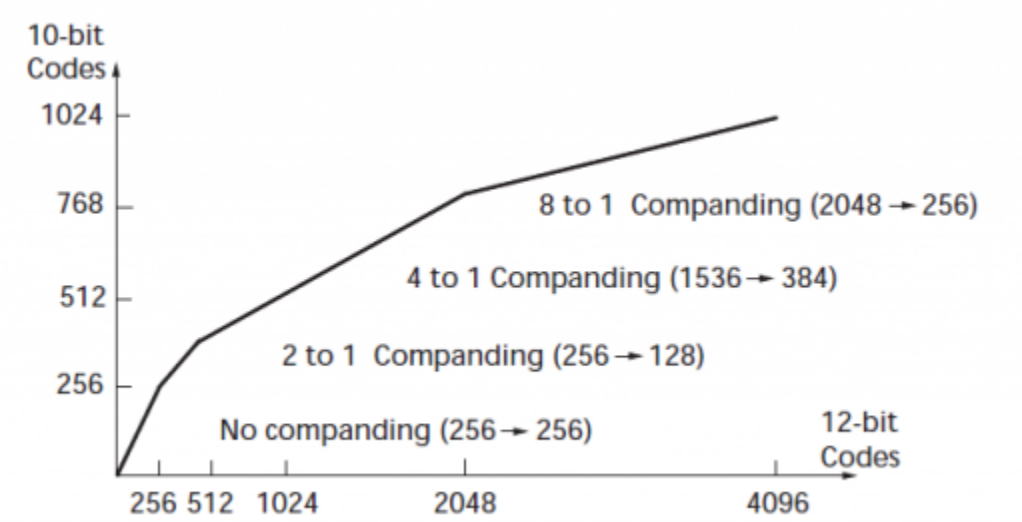
Use `occamSetDeviceValuei(device,OCCAM_ADC_VREF,VALUE)` to change this value. The possible values can be queried with occamGetDeviceValueiv.

7.1.7 ADC Companding



Use `occamSetDeviceValuei(device,OCCAM_ADC_COMPANDING,VALUE)` with `VALUE = 1` or `0` to enable or disable ADC companding. ADC companding is a sensor feature available on the MT9V022 sensors that allows compression of the 12-bit signal into 10-bits (which is then followed by firmware truncation to 8-bits). It has the effect similar to increasing analog gain.

The compression from 12-bit to 10-bit follows the following model:



7.2 Image Processing Controls

7.2.1 Enabling image processing

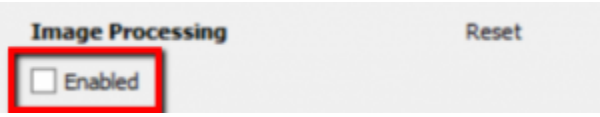
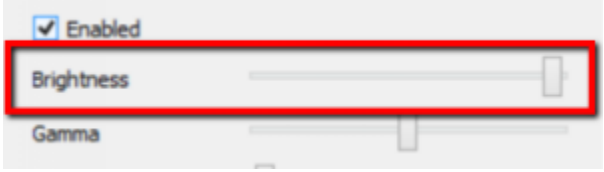


Image processing refers to brightness, gamma, black level, and white balance control. On the Omni series it is computed on the host-side using a lookup-table pixel mapping approach. When this option is enabled, the outputs IMAGE# are processed versions of their counterparts RAW_IMAGE#. When it is disabled, the RAW_IMAGE# images are identical to the IMAGE# images. The remainder of the vision pipeline is performed on the IMAGE# images.

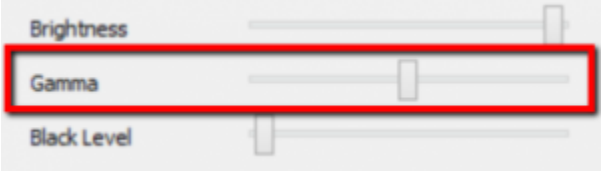
Use `occamSetDeviceValuei(OCCAM_IMAGE_PROCESSING_ENABLED,VALUE)` with `VALUE = 1` or `0` to change this value.

7.2.2 Brightness



This controls the fixed scale factor (digital gain) applied the entire image. The units are scale * 1000. Use `occamSetDeviceValuei(device,OCCAM_BRIGHTNESS,VALUE)` to change this value.

7.2.3 Gamma



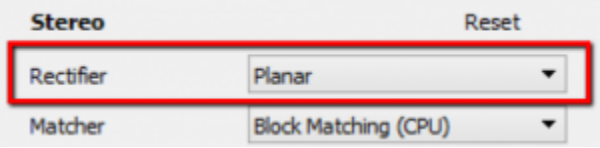
This controls the gamma function applied the entire image. The units are exponent * 1000. Use `occamSetDeviceValuei(device,OCCAM_GAMMA,VALUE)` to change this value.

7.2.4 Black Level

This controls the fixed bias subtracted from the image used to control black level. Use `occamSetDeviceValuei(device,OCCAM_BLACK_LEVEL,VALUE)` to change this value.

7.3 Stereo Controls

7.3.1 Rectifier



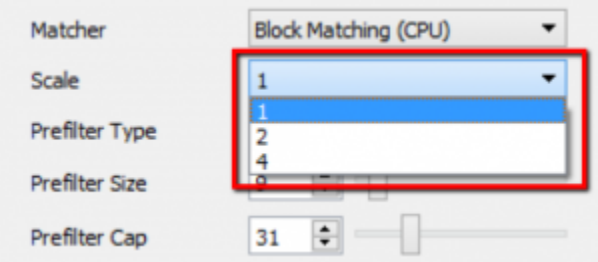
This value controls the stereo rectification engine and is fixed in Omni Stereo to planar rectification.

7.3.2 Matcher

This setting controls which host-side module is responsible for performing stereo matching between the 5 stereo pairs in the camera. Use `occamSetDeviceValues(device,OCCAM_STEREO_MATCHER0,VALUE)` to change this value. The currently supported stereo matchers are:

bmcpu – Block Matching (CPU).

7.3.3 Scale

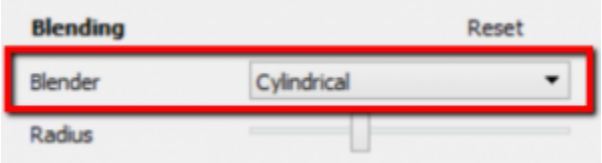


Scale refers to the downsample factor applied to all the images as part of the rectification process. Increasing this value reduces the amount of work and can increase frame rate, but at the cost of reducing resolution.

Use `occamSetDeviceValuei(device,OCCAM_RECTIFY_SCALE,value)` to change this value. You can query available values by using `occamGetDeviceValueiv`.

7.4 Blending Controls

7.4.1 Blender



This value controls which blender module is enabled. This module is used to generate the various stitched outputs such as `OCCAM_STITCHED_IMAGE0`, `OCCAM_STITCHED_DISPARITY_IMAGE`, and so on. Use `occamSetDeviceValues(device,OCCAM_BLENDER0,VALUE)` to change this value. The currently supported blender modules are:

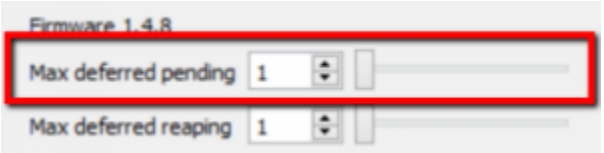
7.5 Advanced Controls

7.5.1 Firmware version

The firmware version is read from the camera and displayed here. It can be read using `occamGetDeviceValuei(device,OCCAM_FIRMWARE_VERSION_A,&value)`, `occamGetDeviceValuei(device,OCCAM_FIRMWARE_VERSION_B,&value)`, `occamGetDeviceValuei(device,OCCAM_FIRMWARE_VERSION_C,&value)`, and `occamGetDeviceValuei(device,OCCAM_FIRMWARE_VERSION,&value)`. The first three values read the A.B.C values separately, and the last value reads a single value where the three values are packed into the lower 3 octets.

On paired devices representing multiple devices, the firmware of all sensors is generally required to be put to the same version. If for whatever reason this is not the case, one of the values will be 255.

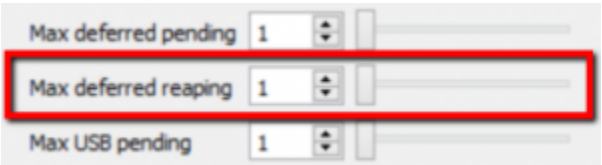
7.5.2 Max Deferred Pending



This value indicates the number of frames that may be queued (but not in process) after reading from the wire, pending host-side processing. For minimal-lag operation, use 1. If the host is not fast enough to process all frames from the camera, frames will be dropped. When a new frame is read from the wire, the oldest frames in the queue are removed until the pending count (including the new frame) is less or equal to this value.

Use `occamGetDeviceValuei(device,OCCAM_MAX_DEFERRED_PENDING_FRAMES,&value)` to read this value.

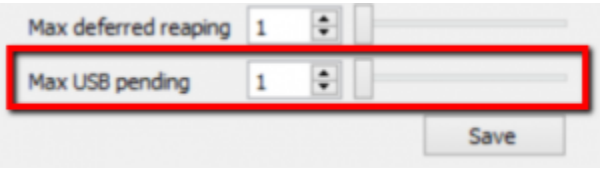
7.5.3 Max Deferred Reaping



This value indicates the number of frames that may be in-process after reading from the wire, pending host-side processing. For minimal-lag operation, use 1. The advantage of setting this value to a value greater than 1 is that depending on what host-side operation is configured, there may be greater overlap at the start/end of processing for a frame.

Use `occamGetDeviceValuei(device,OCCAM_MAX_DEFERRED_REAPING_FRAMES,&value)` to read this value.

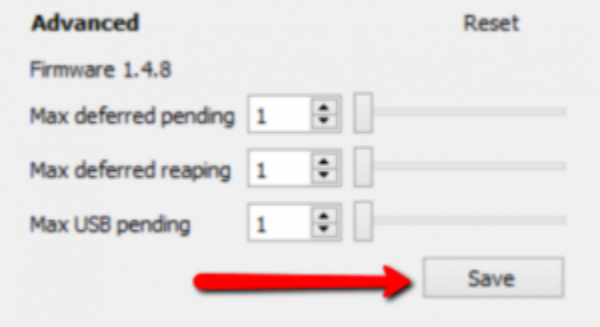
7.5.4 Max USB Pending



The value indicates the number of frames kept in the intermittent queue of USB read frames. It is redundant with the above two settings and will be removed in Indigo SDK 2.1.

Use `occamGetDeviceValuei(device,OCCAM_MAX_USB_PENDING_FRAMES,&value)` to read this value.

7.5.5 Saving Settings



Once you have configured the camera as you wish, you may click Save as indicated above to write all settings to non-volatile EEPROM-memory on the device. On camera and application startup, the driver will automatically read the last-saved settings.

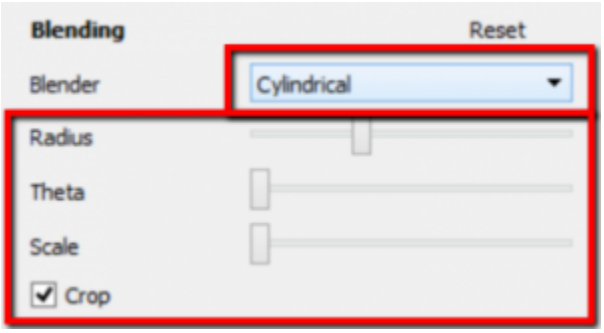
7.6 Cylindrical Blender Controls

This section covers the options available for the Cylindrical blender module, which forms a single image from all the sensors in the camera. The image is formed by projecting each of the sensor images onto a cylinder centered on the Y-axis, and quantizing to a fixed grid to form a 2D image of the unwrapped cylinder. This of course requires full calibration data for the sensors, so their position is known relative to the cylinder.

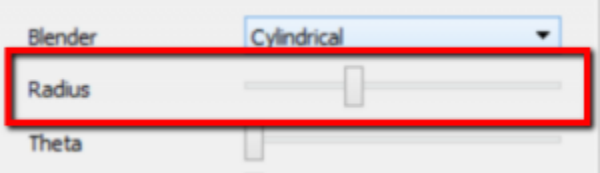
When the camera is plugged in, you can click

to open the camera configuration. The Blending section will show the cylindrical blender options when “Cylindrical” is selected as the Blender.

You can use use `occamSetDeviceValues(device,OCCAM_BLENDER0,"cylb")` to programmatically enable cylindrical stitching if it is not already set. The actual output image will be produced when reading `OCCAM_STITCHED_IMAGE0`, `OCCAM_STITCHED_IMAGE1`, `OCCAM_STITCHED_DISPARITY_IMAGE`, etc. The set of outputs exposed that use the blender depends on the camera/driver.



7.6.1 Radius



The radius option controls the radius of the cylinder that is drawn about the Y-axis. The main affect of this setting is to change parallax errors for objects that are closer or farther from the camera (objects landing exactly on the cylinder will have no blending errors). For objects past several meters, a very large radius should lead to the fewest blending errors. The units are in millimeters * 1000.

Use `occamSetDeviceValuei(device,OCCAM_STITCHING_RADIUS,VALUE)` to change this value.

7.6.2 Theta

This value controls the position (angle) on the cylinder that is used as the started of the unwrapping. It varies from 0 to 360 degrees, and changes where the left edge of the resulting image corresponds to on the cylinder.

Use `occamSetDeviceValuei(device,OCCAM_STITCHING_ROTATION,VALUE)` to change this value.

7.6.3 Scale

The scale option changes the width of the resulting image relative to the height. The units are in scale * 1000, where scale of 2 means width = original_width/2, height = original_height.

Use `occamSetDeviceValuei(device,OCCAM_STITCHING_SCALEWIDTH,VALUE)` to change this value.

7.6.4 Crop

The crop option should be zero or one, and indicates whether to crop the resulting image so that there are no jagged edges on the top/bottom of the image. Typically sensors are not exactly aligned on the X-Z plane, which leads to a trade-off of wasted image vs jagged edges.

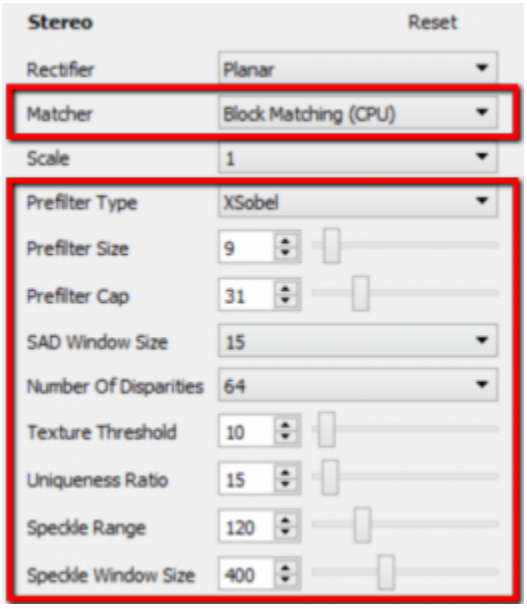
Use `occamSetDeviceValuei(device,OCCAM_STITCHING_CROP,VALUE)` to change this value.

7.7 Block Matching Controls

This section covers the configuration options for the CPU Block Matching module. The module is enabled for Occam stereo rigs such as Omni Stereo. From Indigo Tools, the control pull-down that appears when clicking

will include a section for these options when the stereo matcher module is configured to "Block Matching (CPU)".

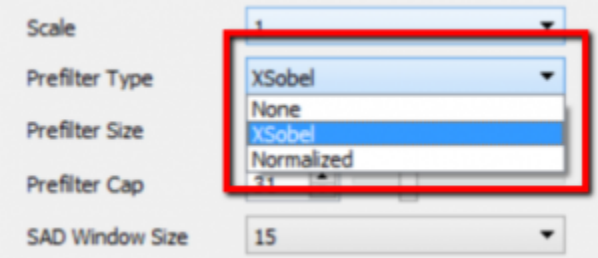
You can use `occamSetDeviceValues(device,OCCAM_STEREO_MATCHER0,"bmcpu")` to enable this module programmatically if it is not already set.



CPU Block Matching is derived from the OpenCV block matching stereo implementation which performs a very efficient SAD cost minimization using SSE instructions on x86 or native C code on other platforms. On nice i7 laptop hardware, performance of around 20 FPS can be achieved computing stereo on all 5 Omni Stereo pairs.

For each of the options described below, programmatic control is possible via the OCCAM_* options listed in the corresponding section, by calling `occamSetDeviceValuei` function. The Indigo Tools application is built on top of the Indigo SDK, so in general anything that is possible to configure with the UI is also possible to do in code using the SDK.

7.7.1 Prefilter Type



Use `occamSetDeviceValuei(device,OCCAM_BM_PREFILTER_TYPE,VALUE)` to change this setting, with the value one of:

```
OCCAM_PREFILTER_NONE = 0
OCCAM_PREFILTER_XSOBEL = 1
OCCAM_PREFILTER_NORMALIZED_RESPONSE = 2
```

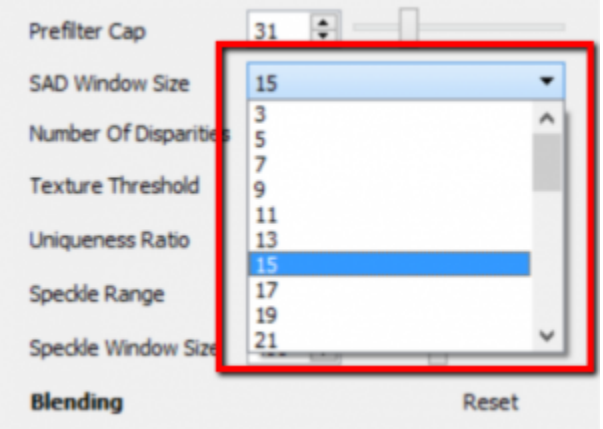
7.7.2 Prefilter Size

Use `occamSetDeviceValuei(device,OCCAM_BM_PREFILTER_SIZE,VALUE)` to change this setting.

7.7.3 Prefilter Cap

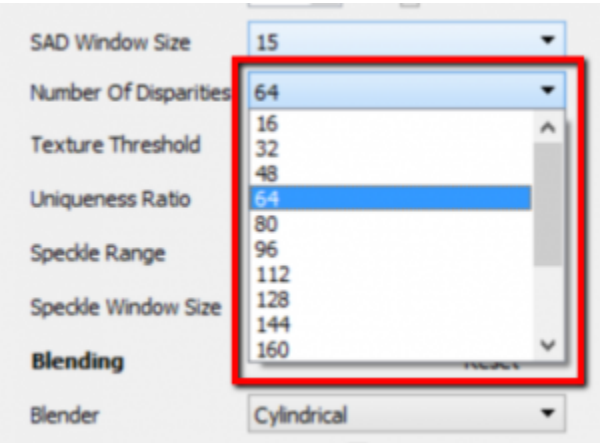
Use `occamSetDeviceValuei(device,OCCAM_BM_PREFILTER_CAP,VALUE)` to change this setting.

7.7.4 SAD Window Size



Use `occamSetDeviceValuei(device,OCCAM_BM_SAD_WINDOW_SIZE,VALUE)` to change this setting.

7.7.5 Number of Disparities



Use `occamSetDeviceValuei(device,OCCAM_BM_NUM_DISPARITIES,VALUE)` to change this setting.

7.7.6 Texture Threshold

Use `occamSetDeviceValuei(device,OCCAM_BM_TEXTURE_THRESHOLD,VALUE)` to change this setting.

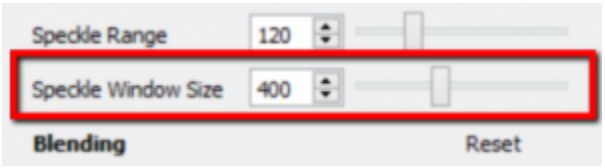
7.7.7 Uniqueness Ratio

Use `occamSetDeviceValuei(device,OCCAM_BM_UNIQUENESS_RATIO,VALUE)` to change this setting.

7.7.8 Speckle Range

Use `occamSetDeviceValuei(device,OCCAM_BM_SPECKLE_RANGE,VALUE)` to change this setting.

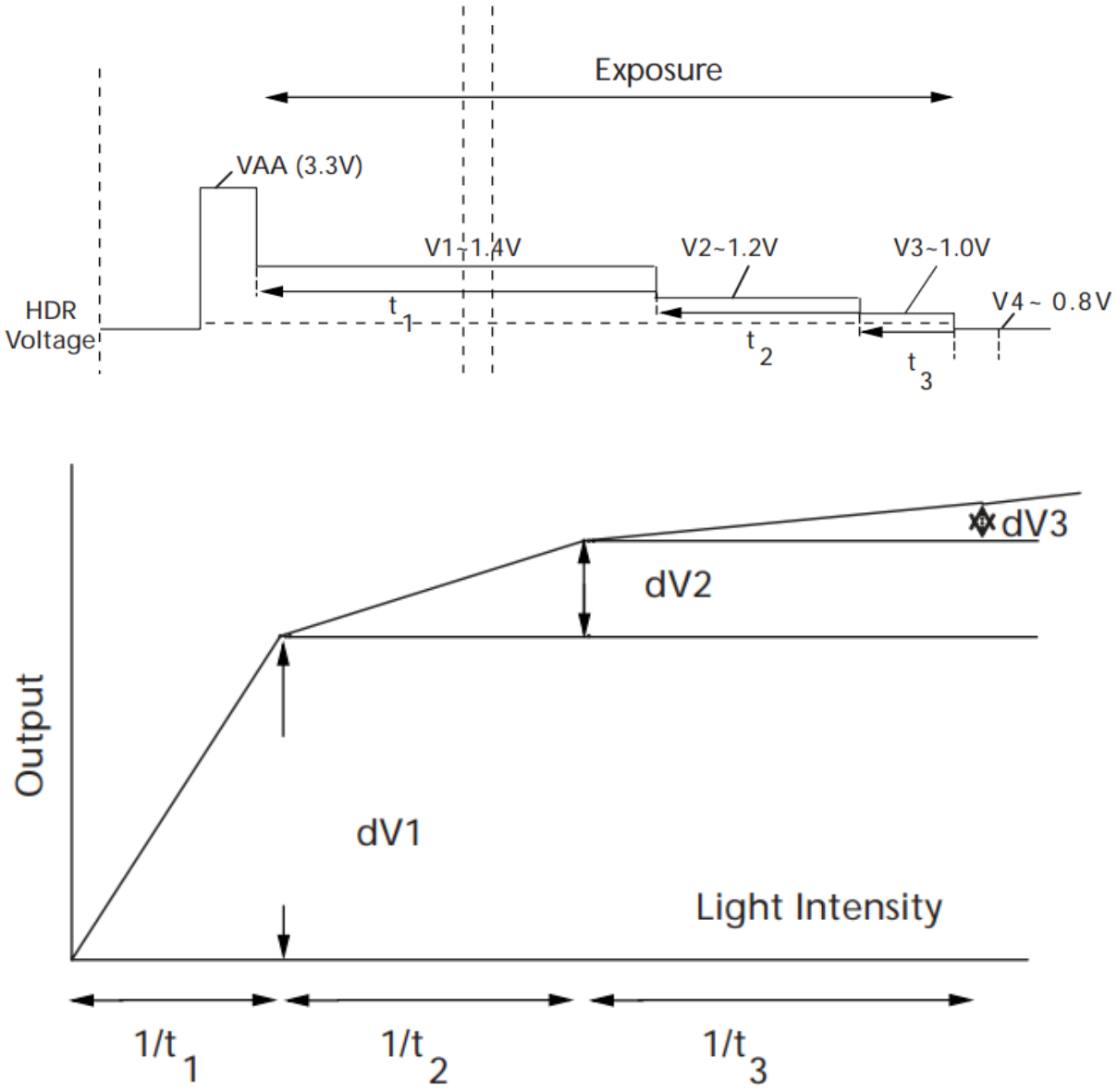
7.7.9 Speckle Window Size



Use `occamSetDeviceValuei(device,OCCAM_BM_SPECKLE_WINDOW_SIZE,VALUE)` to change this setting.

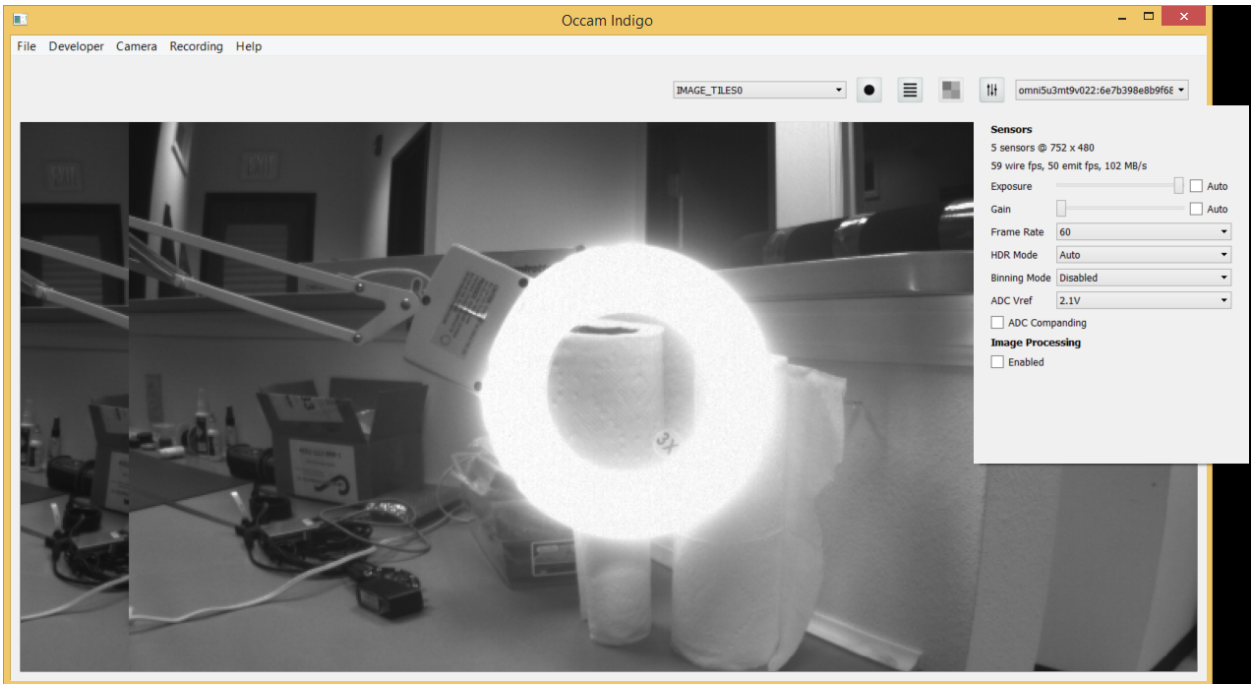
7.8 High Dynamic Range Mode

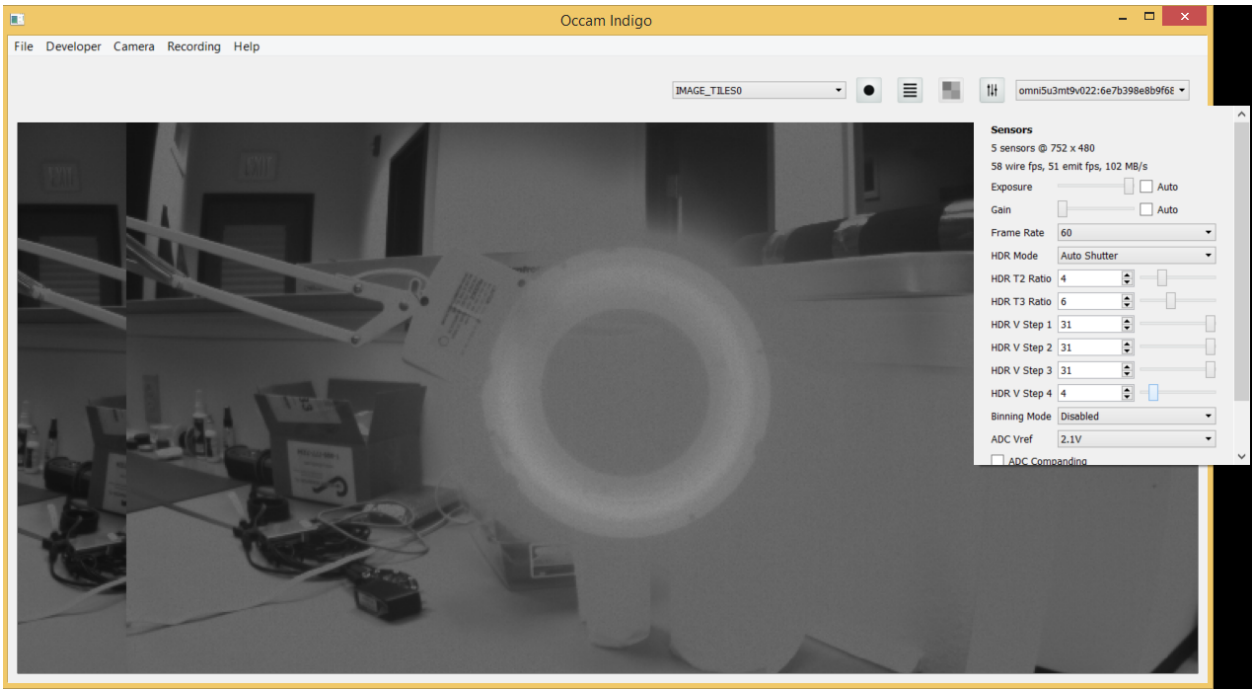
The MT9V022 sensor offers a high dynamic range capture mode, which enables controlling the saturation point of pixels according to exposure time. The mode is enabled by setting the HDR mode (setting `OCCAM_HDR_MODE`) to one of the following: `OCCAM_HDR_MODE_DISABLED`, `OCCAM_HDR_MODE_AUTO_SHUTTER`, `OCCAM_HDR_MODE_MANUAL`. Setting the HDR mode to auto shutter or manual shutter mode enables HDR.

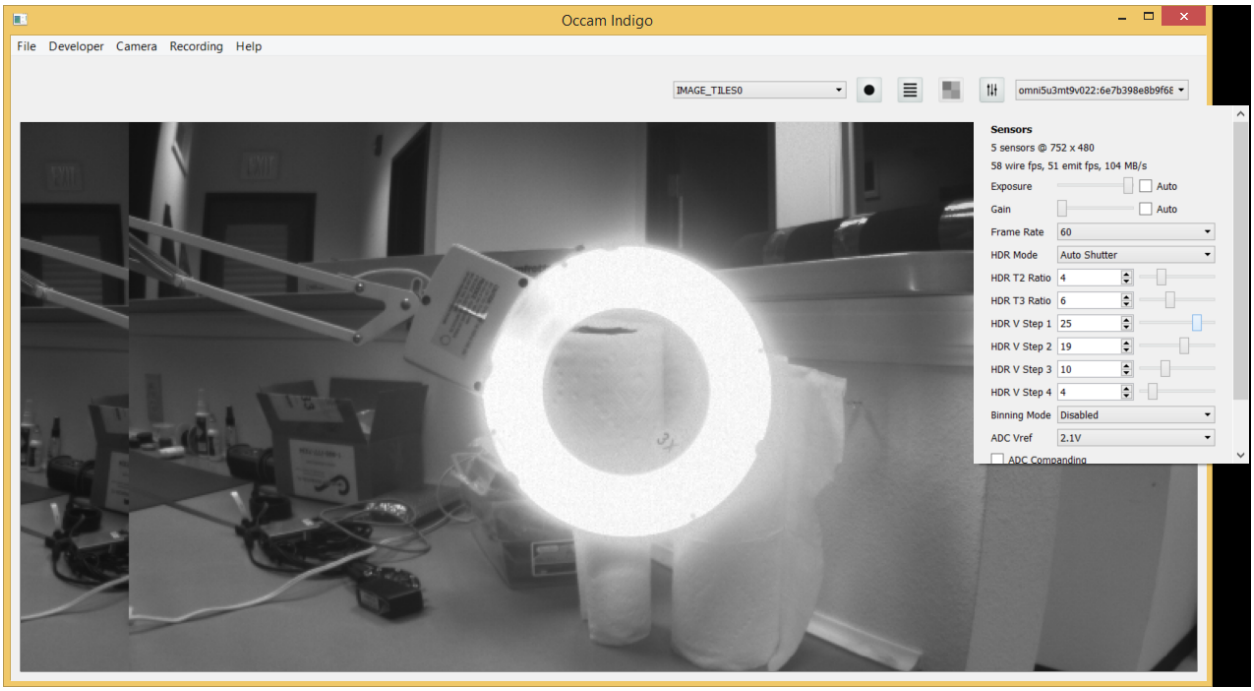


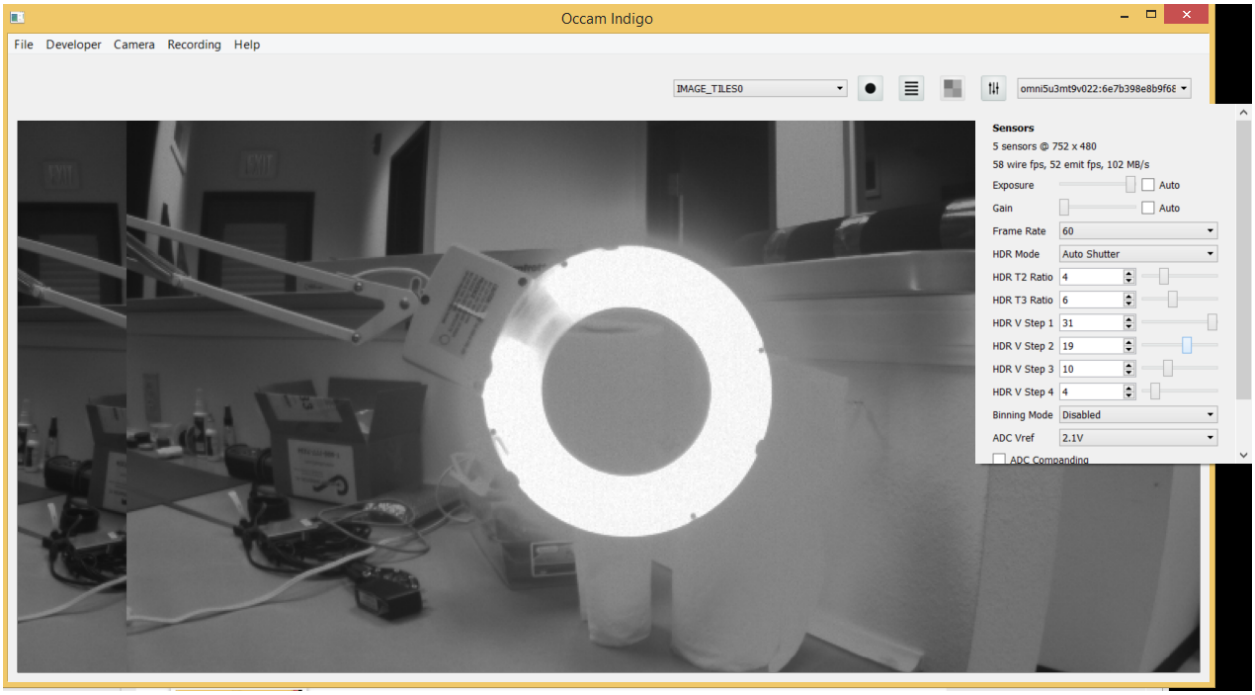
Auto mode refers to determining the time period of the exposure graph, and can be set manually using the `OCCAM_HDR_SHUTTER1` and `OCCAM_HDR_SHUTTER2` settings, or as a function of the exposure time using the settings `OCCAM_HDR_T2_RATIO` and `OCCAM_HDR_T3_RATIO`.

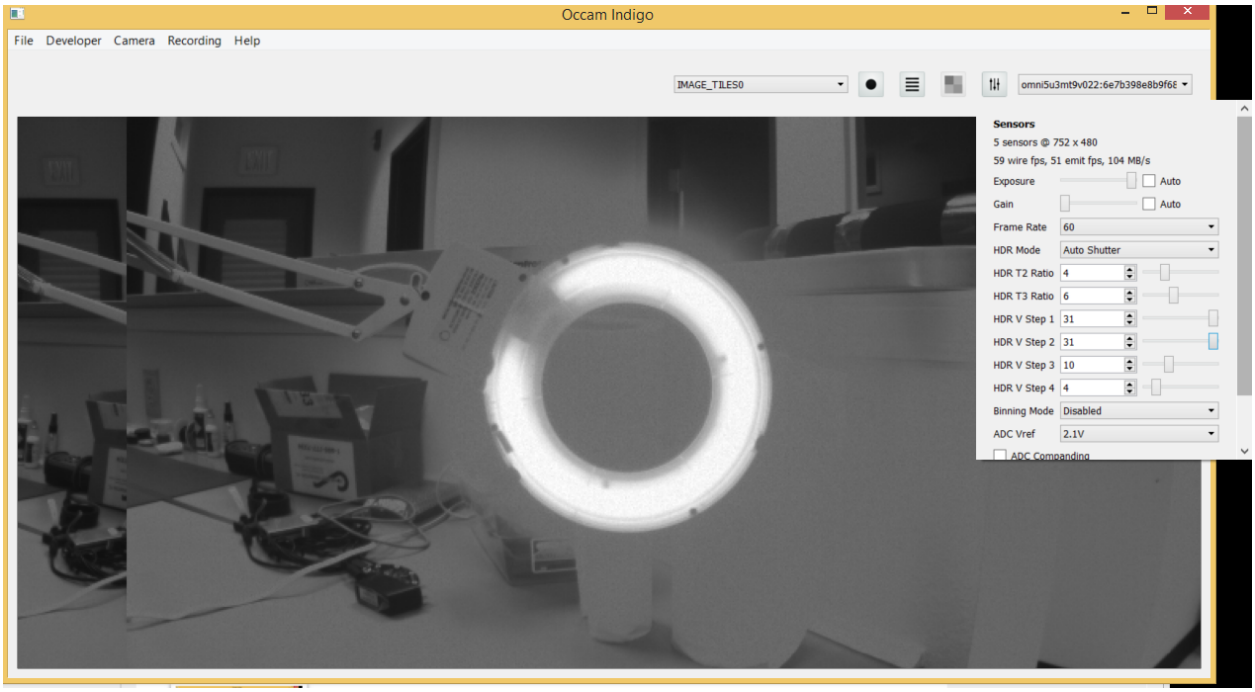
The parameters `OCCAM_HDR_VSTEP1` through `OCCAM_HDR_VSTEP4` effect the knee points in the graph above, which along with shutter control allow control over the graph above. Below are images showing expected response to saturation in light of various settings:


Occam Indigo
File Developer Camera Recording Help
IMAGE_TILES0
Sensors
5 sensors @ 752 x 480
59 wire fps, 50 emit fps, 102 MB/s
Exposure
Auto
Gain
Auto
Frame Rate
60
HDR Mode
Auto
Binning Mode
Disabled
ADC Vref
2.1V
ADC Companding
Image Processing
Enabled


Occam Indigo
File Developer Camera Recording Help
IMAGE_TILES0
Sensors
5 sensors @ 752 x 480
58 wire fps, 51 emit fps, 102 MB/s
Exposure
Auto
Gain
Auto
Frame Rate
60
HDR Mode
Auto Shutter
HDR T2 Ratio
4
HDR T3 Ratio
6
HDR V Step 1
31
HDR V Step 2
31
HDR V Step 3
31
HDR V Step 4
4
Binning Mode
Disabled
ADC Vref
2.1V
ADC Companding

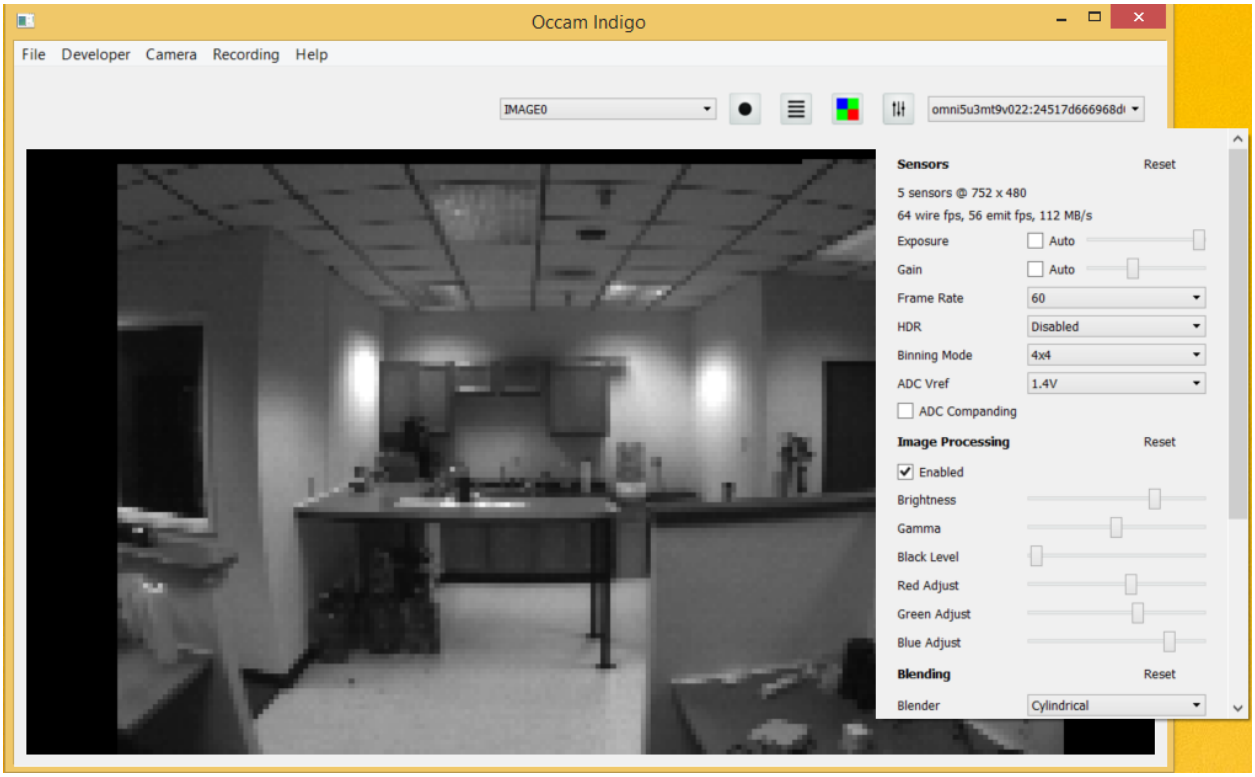

Occam Indigo
File Developer Camera Recording Help
IMAGE_TILES0
Sensors
5 sensors @ 752 x 480
58 wire fps, 51 emit fps, 104 MB/s
Exposure
Auto
Gain
Auto
Frame Rate
60
HDR Mode
Auto Shutter
HDR T2 Ratio
4
HDR T3 Ratio
6
HDR V Step 1
25
HDR V Step 2
19
HDR V Step 3
10
HDR V Step 4
4
Binning Mode
Disabled
ADC Vref
2.1V
ADC Companding

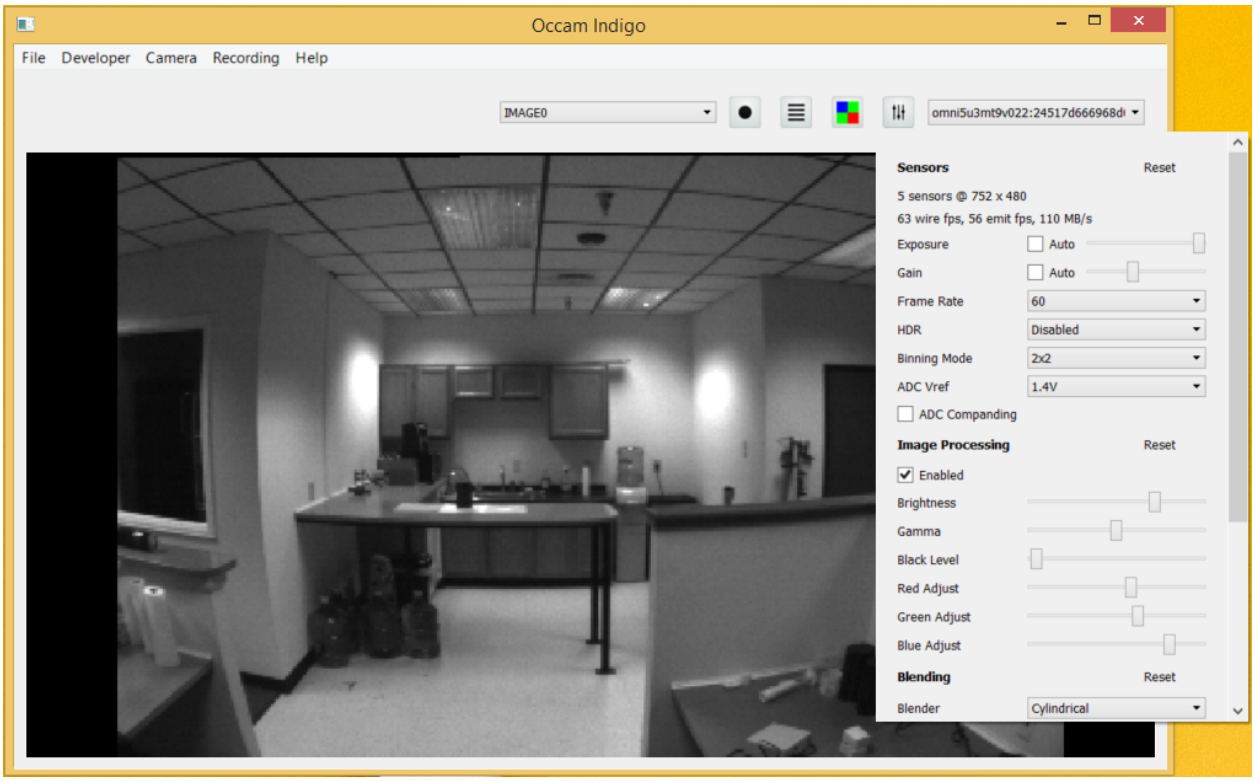

Occam Indigo
File Developer Camera Recording Help
IMAGE_TILES0
Sensors
5 sensors @ 752 x 480
58 wire fps, 52 emit fps, 102 MB/s
Exposure
Auto
Gain
Auto
Frame Rate
60
HDR Mode
Auto Shutter
HDR T2 Ratio
4
HDR T3 Ratio
6
HDR V Step 1
31
HDR V Step 2
19
HDR V Step 3
10
HDR V Step 4
4
Binning Mode
Disabled
ADC Vref
2.1V
ADC Companding

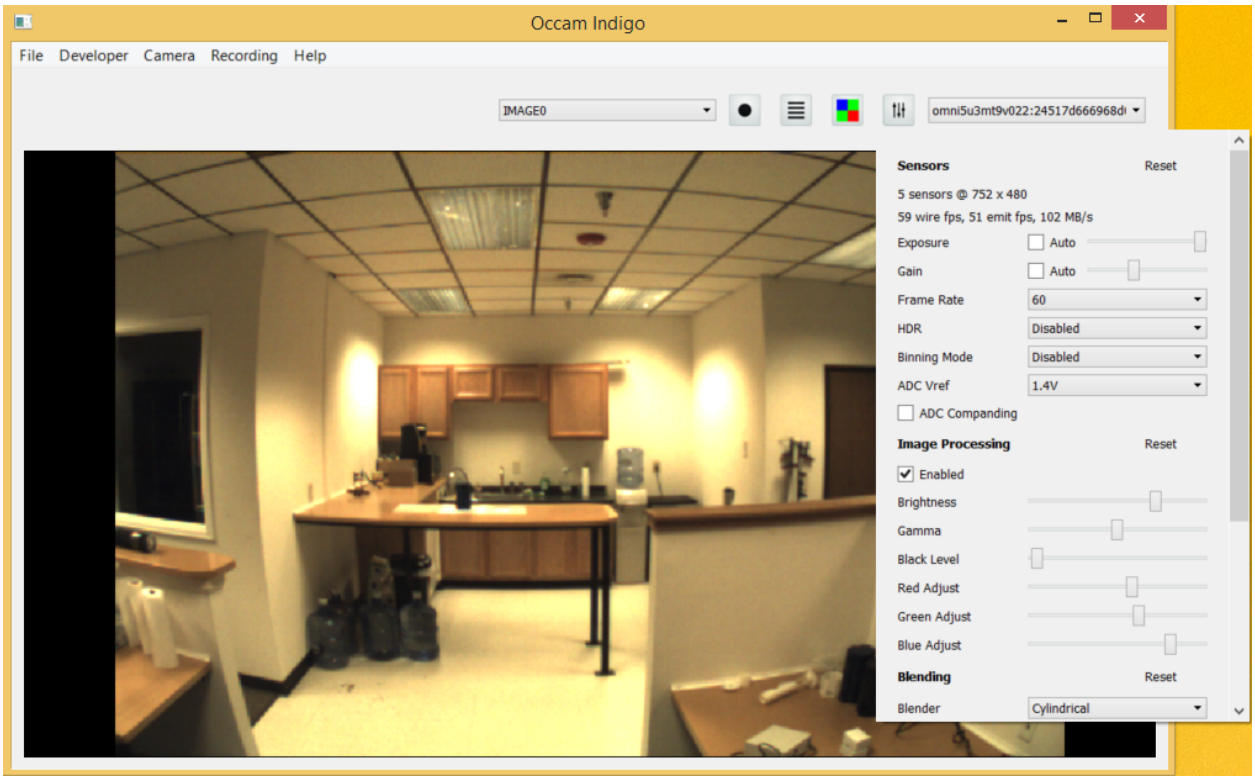


7.9 Binning Mode

The MT9V022 sensor supports binning mode which downsamples the image by averaging blocks of 2x2 or 4x4 pixels in hardware. Binning mode can be controlled by changing the setting `OCCAM_BINNING_MODE` to one of `OCCAM_BINNING_DISABLED`, `OCCAM_BINNING_2x2`, `OCCAM_BINNING_4x4`. Below are examples of these three binning modes:


Occam Indigo
File Developer Camera Recording Help
IMAGE0
omni5u3mt9v022:24517d666968d
Sensors
Reset
5 sensors @ 752 x 480
64 wire fps, 56 emit fps, 112 MB/s
Exposure
Auto
Gain
Auto
Frame Rate
60
HDR
Disabled
Binning Mode
4x4
ADC Vref
1.4V
ADC Companding
Image Processing
Reset
Enabled
Brightness
Gamma
Black Level
Red Adjust
Green Adjust
Blue Adjust
Blending
Reset
Blender
Cylindrical


Occam Indigo
File Developer Camera Recording Help
IMAGE0
omni5u3mt9v022:24517d666968d
Sensors
Reset
5 sensors @ 752 x 480
63 wire fps, 56 emit fps, 110 MB/s
Exposure
Auto
Gain
Auto
Frame Rate
60
HDR
Disabled
Binning Mode
2x2
ADC Vref
1.4V
ADC Companding
Image Processing
Reset
Enabled
Brightness
Gamma
Black Level
Red Adjust
Green Adjust
Blue Adjust
Blending
Reset
Blender
Cylindrical



7.10 Device Outputs

7.10.1 Raw hardware output

For both Omni 60 and Omni Stereo, the actual output from the hardware are raw images. These are either intensity images for the monochrome versions of the camera, or bayer pattern images for the color versions of the camera.

The remainder of the pipeline occurs on the host and is performed by the SDK: the debayer processing to convert bayer pattern to RGB, image processing (which includes gamma correction, brightness, white balance), rectification, stereo processing, and stitching.

The SDK is designed such that you can request any stages of the pipeline using the `occamDeviceReadData` API, and the SDK will perform only the processing required to produce the requested outputs. For example, if only the raw image outputs are requested, then no work will happen on the host beyond receiving the data from the hardware. If a single or a subset of stereo pairs are requested, then only the color and stereo processing needed to produce that subset will be performed. If full output is requested (such as colored point cloud and stitched image), then the entire pipeline will be run. The SDK uses multiple threads to utilize as many cores as there are available.

7.10.2 Processed outputs

OCCAM_IMAGE_TILES0

This is the set of sub-images (one per sensor) tiled in a grid.

OCCAM_RAW_IMAGE_TILES0

This is the set of sub-images (one per sensor) tiled in a grid, but in raw format. This means the bayer pattern should be visible, and no brightness/gamma or other correction will have been applied.

OCCAM_IMAGE_TILES1

For the Omni Stereo camera, this emits tiled images for the top 5 sensors of the camera, on top of the tiled heatmap depth images computed using both the top 5 and bottom 5 sensors.

OCCAM_TILED_DISPARITY_IMAGE

For the Omni Stereo camera, this emits the tiled bottom row depth images.

OCCAM_STITCHED_IMAGE0

This shows the 360-degree cylindrically stitched representation from all 5 sensors. For the Omni Stereo, this uses the top row of sensors.

OCCAM_STITCHED_IMAGE1

For the Omni Stereo, this emits the 360-degree cylindrically stitched representation from the top 5 sensors, as well as the cylindrically stitched depth image computed from both the top and bottom sensors.

OCCAM_STITCHED_DISPARITY_IMAGE

For Omni Stereo, this emits the 360-degree cylindrically stitched depth image, computed from both top and bottom sensors.

OCCAM_IMAGE0 – OCCAM_IMAGE9

These outputs emit the individual-sensor image outputs. For Omni 60, OCCAM_IMAGE0 – OCCAM_IMAGE4 are valid (for sensors 1 to 5). For Omni Stereo, OCCAM_IMAGE0 – OCCAM_IMAGE9 are valid (for sensors 1 to 10).

OCCAM_RAW_IMAGE0 – OCCAM_RAW_IMAGE9

These outputs emit the individual-sensor image outputs, in raw format. This means the bayer pattern should be visible, and no brightness/gamma or other correction will have been applied.

For Omni 60, OCCAM_IMAGE0 – OCCAM_IMAGE4 are valid (for sensors 1 to 5). For Omni Stereo, OCCAM_IMAGE0 – OCCAM_IMAGE9 are valid (for sensors 1 to 10).

OCCAM_RECTIFIED_IMAGE0 – OCCAM_RECTIFIED_IMAGE9

For Omni Stereo, this emits the individual-sensor rectified images. This is standard planar rectification that warps the image such that epipolar lines are straightened and stereo matching can proceed linearly. Because the stereo configuration is vertical, the output of this step produces transposed images (480 x 752) that can then be matched left-to-right as is standard for stereo matching algorithms.

OCCAM_DISPARITY_IMAGE0 – OCCAM_DISPARITY_IMAGE4

For Omni Stereo, this returns the immediate output of stereo matching, the disparity/depth images, each stereo pair in the camera. The camera can be thought of as five individual stereo pairs, and this is then the stereo matcher output of each stereo pair.

OCCAM_POINT_CLOUD0 – OCCAM_POINT_CLOUD4

For Omni Stereo, these are the colored point cloud outputs of each stereo pair. The coordinate system of the points are in the same coordinate frame of the entire camera, which is centered in the ring of 5 sensors for Omni 60 and the center of the top ring for Omni Stereo. The units are millimeters.

OCCAM_STITCHED_POINT_CLOUD

For Omni Stereo, this is the combined point cloud from all five stereo pairs.

7.11 Upgrading Device Firmware

This section covers how to upgrade firmware using the Indigo Tools software. Your camera will come with the latest firmware, and in general upgrading firmware is not necessary and may cause problems if interrupted. Unless you know that you need to upgrade, you are likely better off staying where you are.

If your camera is bricked (nonfunctional) after a failed upgrade process, please see the troubleshooting section to restore firmware manually.

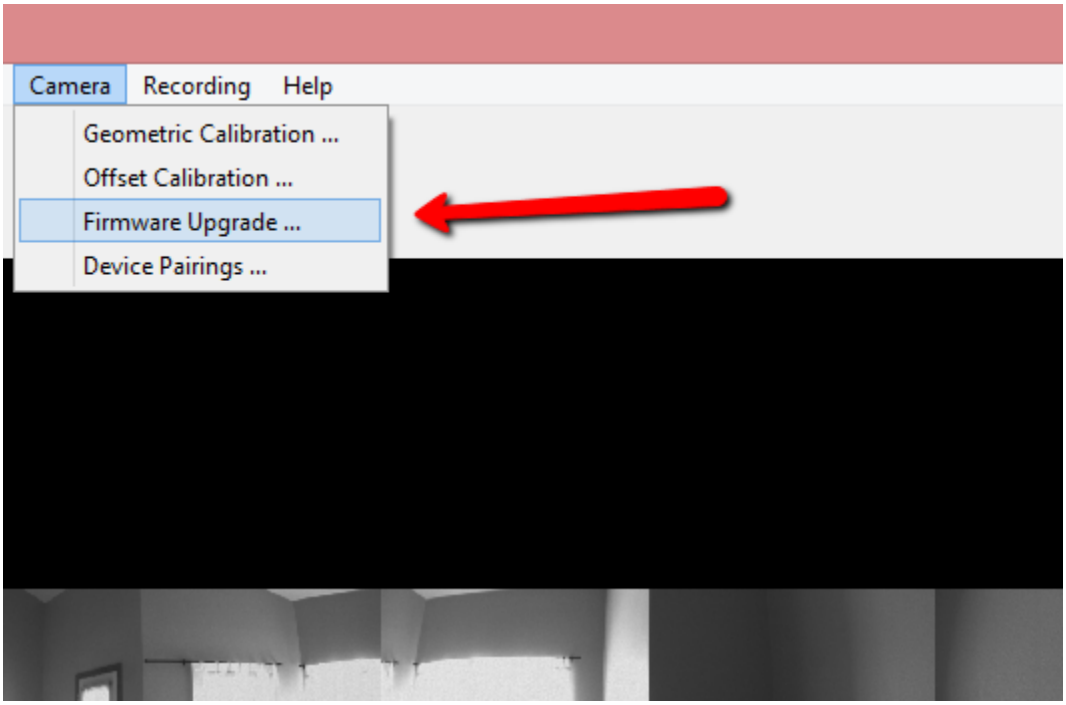
- Step 1:

Before you start the process, close the software and unplug the camera to bring everything to a known state.

Open Indigo Tools, and plug in the camera that you want to upgrade the firmware on.

- Step 2:

Open the firmware upgrade tool.

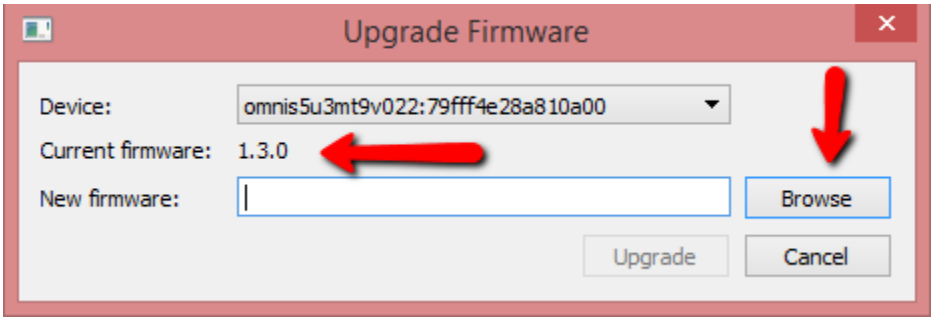


- Step 3:

Make sure the firmware version reads correctly for your camera. The "Current firmware" field should match the version of the firmware you have installed currently. If the value doesn't match or something else doesn't look right, stop and investigate further and possibly contact Occam support before proceeding to flash a new firmware. Flashing the wrong firmware can brick your camera at which point it will require servicing or be non-functional, so you want to proceed with caution.

For paired multi-cameras such as Omni Stereo and multiple Array cameras linked together, the firmware version much match between all the devices. If a device in the paired group does not match or is not actually connected (i.e., the paired device is partially connected– see the paired devices tool for more information), some of the digits may be 255.

In the normal case the screen will appear like:

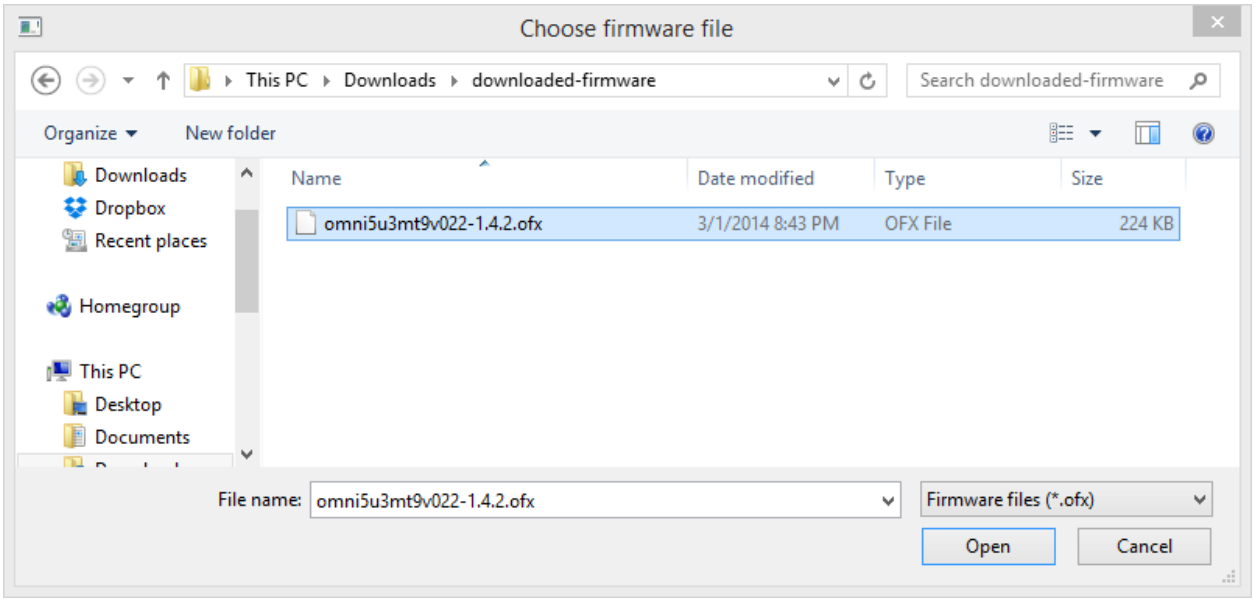


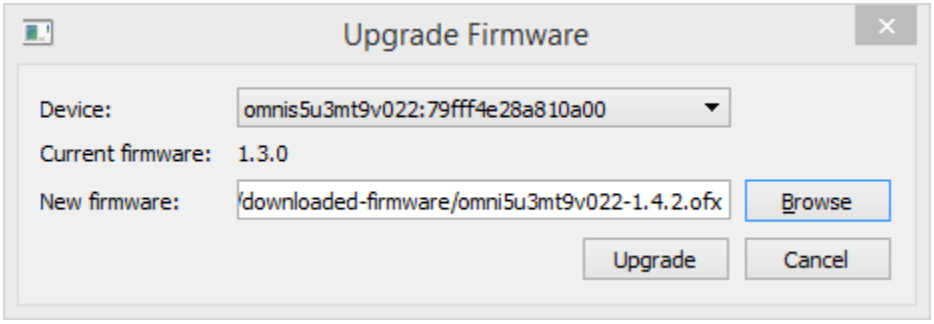
If everything looks correct, click browse to select the firmware image you downloaded from the Occam web site.

- Step 4:

Make sure the model number of the device matches what you have. In the 1.x series of the software it is not prohibited to flash a camera with the firmware for another camera. This may brick your device, depending on the combination (in most cases the device will still be functional/upgradeable, but it will not return images or other data).

For paired devices such as omnis5u3mt9v022 (aka Omni Stereo), the firmware image filename will have the constituent model number omni5u3mt9v022.

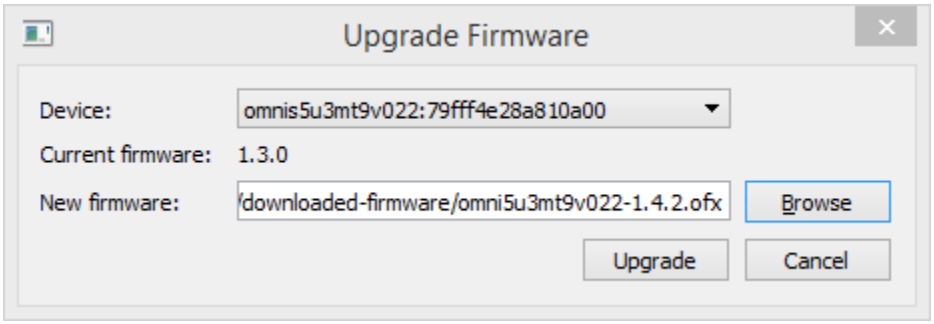




- Step 5:

Once you are absolutely sure everything is correct, click the Upgrade button. During the upgrade process, a progress bar will appear showing the firmware upgrade progress. It is critically important that you don't stop the program while this is happening as the result can be that your device will be nonfunctional afterwards and require servicing. So make sure your laptop is plugged in.

The process should not take more than a couple of minutes. Once it completes, the device will reset and should re-enumerate in Indigo Tools.



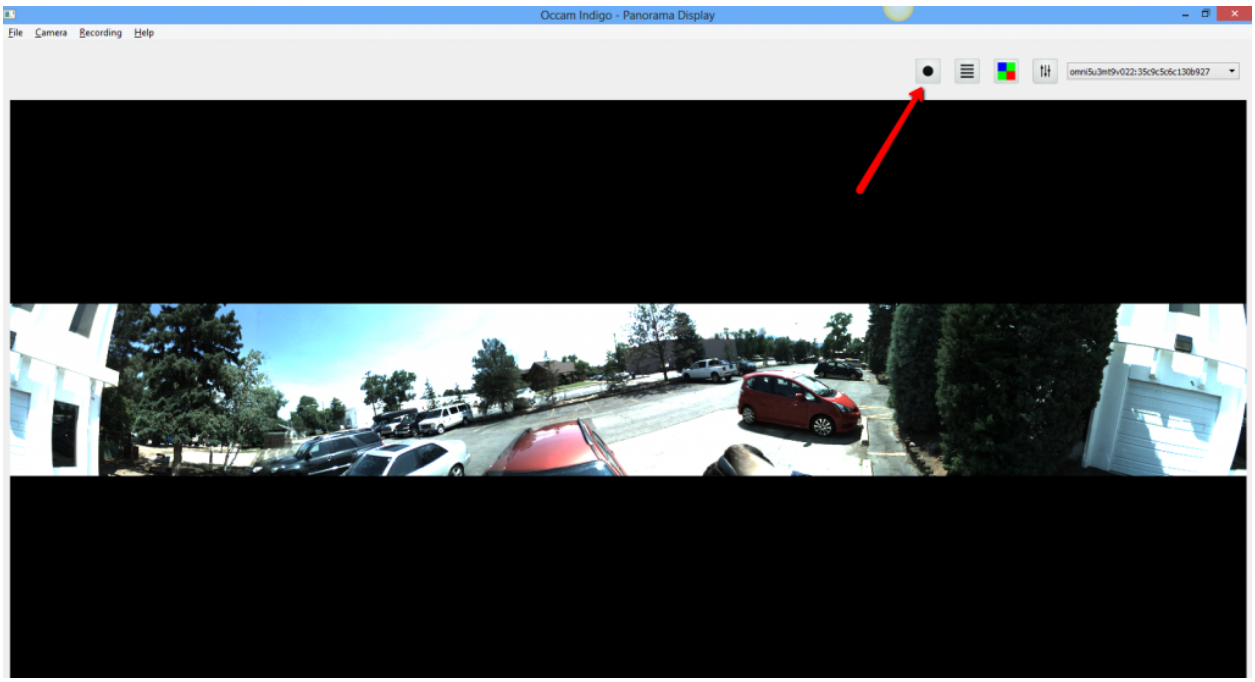
CHAPTER

EIGHT

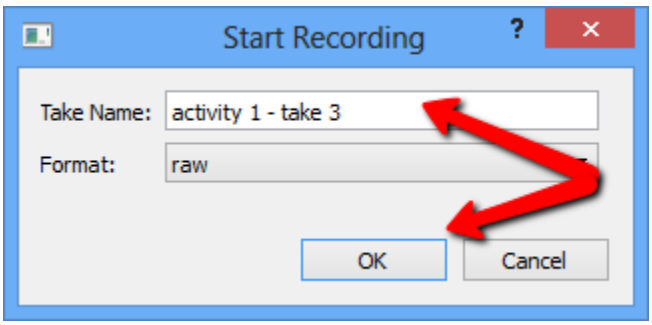
RECORDING

8.1 Recording Video with Indigo Tools

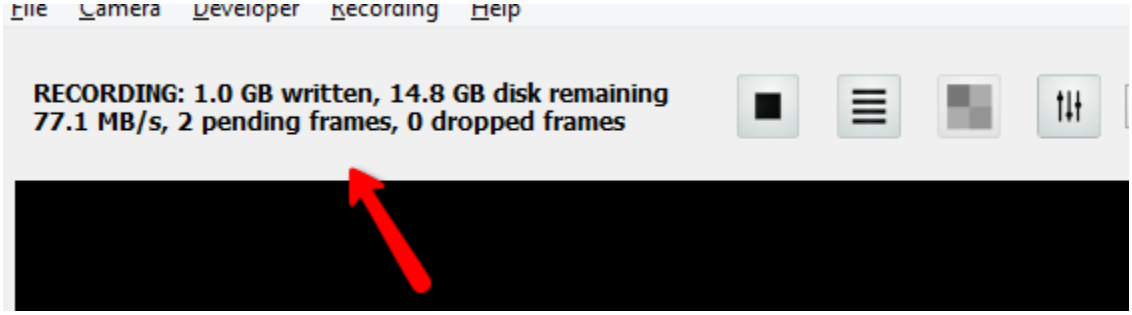
- Step 1: Click the record button on the main screen as indicated in the picture:



- Step 2: The recording options dialog box will appear. Here you can change the name of the "take" (which just refers to the video recording). The take names don't have to be unique. You can use the same name each time if you want, or for a whole series of different takes. To start recording, click the OK button.



- Step 3: The video will now be recording. Information about the recording process is printed in the upper left corner of the main window as the recording takes place.



On the first line, the values are:

The total amount of data written to disk. Note that since this is RAW video, the disk space required to record is fairly substantial. Each frame can be from 1.25 to 1.8 MB, and at 60 FPS that means 70 – 103 MB/s. The frame size varies depending on whether you have stitching enabled. When there is no stitching, the disk space required per second will be (752*480*60*5)/1024/1024=103.2MB/s. When stitching is enabled, it will be slightly less since ~20% of each sensor frame that overlaps with an adjacent sensor frame will be de-duplicated by blending it with its neighbor.

The second value tells you how much disk space is remaining. This refers to the free space available on the disk that holds the recording folder.

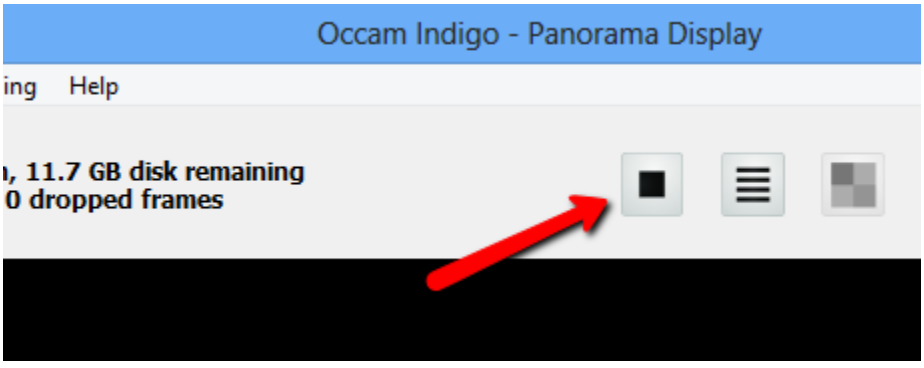
The second line has values that pertain to the measured recording rate and the queueing information:

The first value gives the MB/s that the software is actually writing to disk. Normally this is the same as the camera data rate unless your disk is slower than the camera data rate. For the Omni 60 camera this is unlikely since the maximum rate for RAW bayer will be 103MB where standard disk rates are usually in the 150 – 250 MB/s range.

The second value is the number of frames that are currently queued to be written to disk. If the operating system throttles Indigo Tools because the disk has become busy with something higher priority, then this number will accumulate with the frames that we haven't yet been able to write to disk. In normal working condition this number will generally remain very low and stay low indefinitely. If it climbs too high the software will begin to drop frames.

The last number gives the number of frames dropped. Frames drop are not desirable of course and will mean the recording will be less than 60 FPS.

- Step 4: Click the record button again (it will be a square icon now) to stop the recording.



This will open the video recording as the main screen, and from there you can export, rename, and delete the video. If you recorded from Raw mode, there will also be a color processing pop-up available on this screen to change the demosaicing and color processing options during post.

Click the left arrow button on the upper left of the screen to return to live mode.

8.2 Data rates and color processing

The raw sensor output from the Omni 60 is standard Bayer pattern. The raw output of the color camera looks like this:

Each block of 2 x 2 pixels has two green pixels, one blue pixel, and one red pixel. A process known as demosaicing is required to convert a bayer pattern image to an RGB image. This process is unfortunately not as simple as separating the three colors into three separate color planes, since the data is really a mix of spatial as well as chroma information. On the other hand, bayer pattern gives a very compact way of storing color information in an image without increasing the image size, as compared to other standard pixel formats such as YUV420 which always increase the data size by 1.5x or more (the original luminance image Y, and then downsampled chroma images U and V). For this reason bayer pattern is the standard format that virtually all image sensors use.

The Indigo SDK and Tools provide a few pipelines that dictate where and when the demosaicing happens:

- Raw: demosaicing happens in the display layer, as a post-processing step. Video files contain the bayer pattern data, not the RGB data.
- CPU: demosaicing happens in the camera driver, in CPU. Video files contain the RGB data, not the bayer pattern data.
- OpenGL: demosaicing happens in the camera driver, in OpenGL (GPU). Video files contain the RGB data, not the bayer pattern data.

8.3 Recording Folder

The recording directory is located in the folder

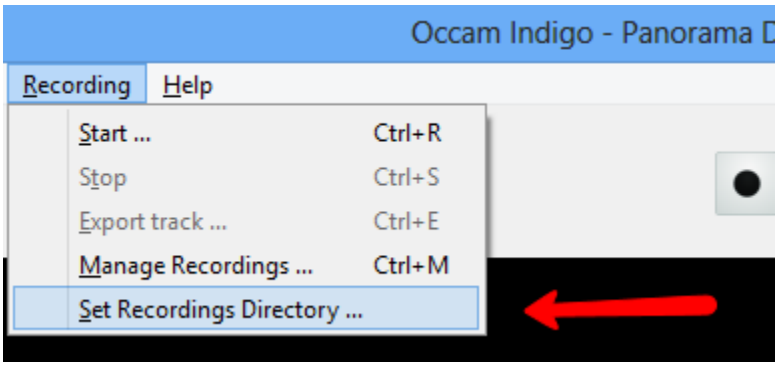
```
<USER_FOLDER>\My Documents\Occam\Recordings
```

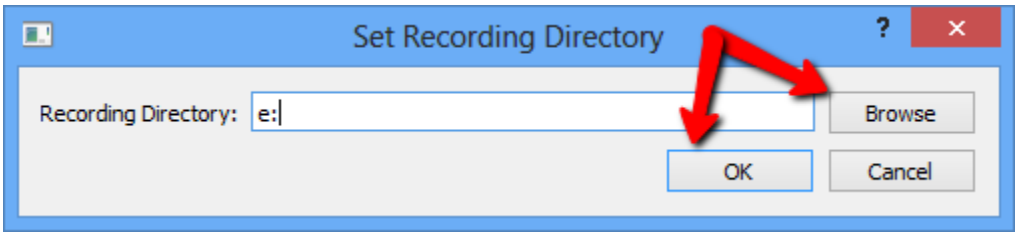
Usually <USER_FOLDER> is something like

```
C:\Users\<USERNAME>
```

where `USERNAME` is your log in name.

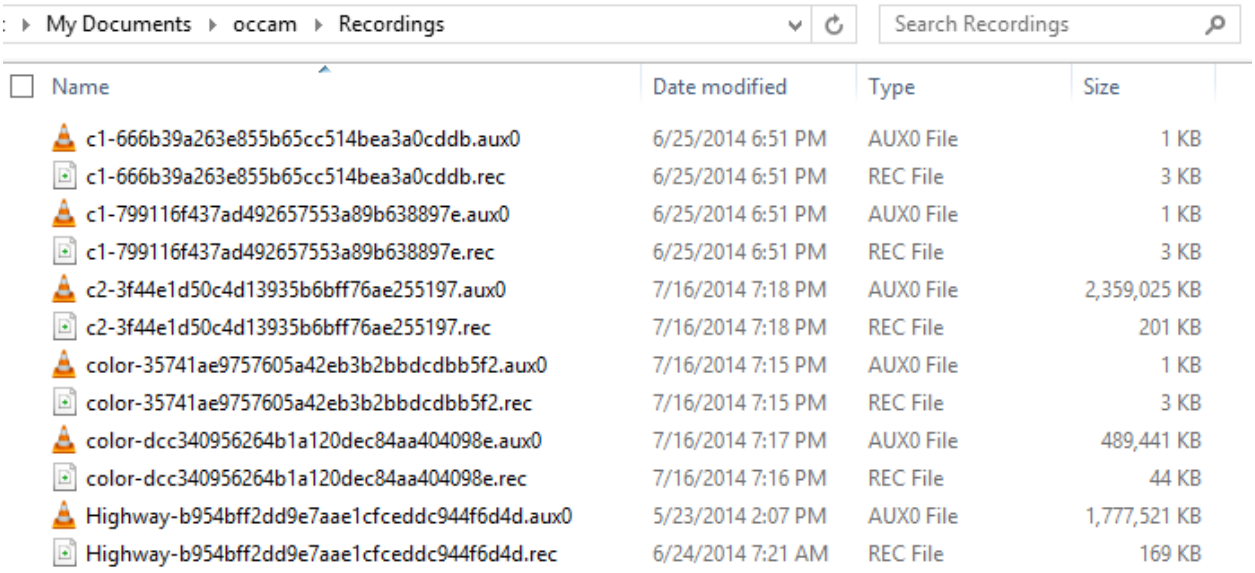
In Indigo Tools 1.3, the recording directory can be changed by going to Recording | Set Recordings Directory in the main menu.





You can manually enter the name or click Browse to select the recording directory via the standard Windows directory select dialog box.

The recording directory will contain .rec and .aux0 files, for example:



The .rec files are a sqlite3-based index file that gives metadata about each frame. The .aux0 files contain the raw video frame data. Each recording as a unique ID as well as the user given name (accessible when starting the recording and via the rename function).

You can copy these files out of the recording directory to move them to other medium or whatever you like, but be sure to keep the files that correspond to a single recording together.

CHAPTER

NINE

USING THE INDIGO SDK

9.1 Overview

The SDK provides programmatic control over the camera(s) connected to a host system and is the basis that Omni cameras are normally used by other projects.

There are a number of examples contained the `examples` top-level folder of the SDK. It is recommended to start by understanding the `read_images.c` and `read_images_opencv.cc` examples first. These perform device enumeration, opening of devices, and streaming of video data. `read_images_opencv.cc` additionally demonstrates how to convert the read images to OpenCV format and display them with OpenCV.

The entire SDK API is described in `include/indigo.h` in the SDK, and the header file contains detailed information about each function and its parameters.

9.2 Initializing the SDK

The first step to using the SDK is calling `occamInitialize`. On shutdown, the `occamShutdown` should also be called.

9.3 Enumerating and Opening Devices

Next, the `occamEnumerateDeviceList` API can be used to enumerate the set of Occam cameras that are present on the system. Each camera is identified with a `cid`, which is a string of the format `model:serial` where model is the full model number of the camera (e.g., omni5u3mt9v022 for Omni 60, and omnis5u3mt9v022 for Omni Stereo), and serial is the unique serial number of the device.

Once you have a cid, you can call `occamOpenDevice` to actually connect to the device. This also starts the host-side processing threads that will do the host-side work as requested later.

9.4 Configuring Devices

To configure device parameters, such as exposure period, gain value, and others, use `occamSetDeviceValuei`, `occamSetDeviceValuer`, `occamSetDeviceValues`, `occamSetDeviceValueiv`, `occamSetDeviceValuerv`, `occamSetDeviceValuesv`. These are the same interface but with different data types. *i* stands for integer values, *r* for real values, *s* for string values, and *v* suffix stands for a vector of values. Most device settings use integer values. Calibration values are typically given as real vectors. Model and serial number are given as strings.

Use the occamGetDeviceValuei, occamGetDeviceValuer, occamGetDeviceValues, occamGetDeviceValueiv, occamGetDeviceValuerv, occamGetDeviceValuesv APIs to retrieve the currently assigned configuration.

Use the occamEnumerateParamList and occamFreeParamList to list the full set of configuration parameters that are available on the device. The full list of device parameters is given by the OccamParam enum in include/indigo.h, though all devices will not support the full set of configuration.

In general, anything that is configurable by the UI in Indigo Tools is configurable using the SDK. The section of this manual that cover the UI interface also indicate which parameters each control is changing under the hood.

9.5 Reading Data

To actually read data from the device, use the occamDeviceReadData API call. This takes an array of output IDs (e.g., OCCAM_IMAGE0, OCCAM_STITCHED_IMAGE0) and returns a set of pointers to SDK objects containing the data. The main two types of data returned are OccamImage and OccamPointCloud. OccamPointCloud is returned for OCCAM_POINT_CLOUD0 through OCCAM_POINT_CLOUD4 as well as OCCAM_STITCHED_POINT_CLOUD, and the remaining outputs all return OccamImage.

The SDK will only perform the minimal subset of the pipeline required to produce the requested outputs.

9.6 Reading calibration data from camera storage

Every Occam camera is equipped with non-volatile EEPROM memory that is used to store calibration data, along with other sensor, vision module, and user settings.

Omni cameras are pre-calibrated at factory time and the calibration will be stored on the camera's memory. Using Indigo Tools you can re-generate the calibration and overwrite the data that is held on the camera.

If the calibration data is not set, it will be returned as canonical values (D=0, K=eye(3), R=eye(3), T=0). For the parts that are non-canonical, the values for all sensors will be within the same coordinate frame. For Omni 60 cameras, the center of the world is in the center of the sensor ring. For Omni Stereo, the center of the world is the center of the top sensor ring.

9.6.1 Device values representing calibration

When the driver for your device loads, it will read the calibration data from non-volatile memory and make it accessible through the following settings that you can read with the occamGetDeviceValuerv() API call.

OCCAM_SENSOR_DISTORTION_COEFS#

OCCAM_SENSOR_INTRINSICS#

OCCAM_SENSOR_ROTATION#

OCCAM_SENSOR_TRANSLATION#

Where the # is replaced with the digit 0, 1, etc, up until the number of sensors on the camera. You can read the

OCCAM_SENSOR_COUNT

setting from the camera to determine the number of sensors. For Omni series this will be fixed to 5 or 10 depending on the configuration.

9.6.2 Calibration format / camera model

The format of the calibration data is the same as is used in OpenCV, and is as follows:

- OCCAM_SENSOR_DISTORTION_COEFS# This is vector D of 5 coefficients indicating radial and tangient coefficients representing the mapping from undistorted to distorted coordinates in normalized camera space (X and Y varying in $[-1, 1]$).
- OCCAM_SENSOR_INTRINSICS# This is the 3×3 matrix K of camera internal parameters in this format: $[fx, 0, cx; 0, fy, cy; 0, 0, 1]$ where fx and fy are the focal length multiplied by the mapping of pixel dimensions to metric dimensions (usually millimeters depending on how the calibration tool is configured), and cx and cy are the principal point of the camera. This matrix maps from normalized camera coordinates (distorted or not) to original image coordinates.
- OCCAM_SENSOR_ROTATION# This is the 3×3 rotation matrix R indicating the rotation part of the mapping from world coordinates to normalized camera space.
- OCCAM_SENSOR_TRANSLATION# This is the 3×1 translation vector T indicating the pre-rotated translation part of the mapping from world coordinates to normalized camera space.

The full camera model used is the same as in OpenCV, and is also the standard Hartley/Zisserman projective camera model:

$$xh = K * D([R; T]X)$$

9.6.3 SDK Example

Please refer to the examples/read_calib.cc example in the Indigo SDK for a more verbose example of how to read calibration data from a camera.

9.6.4 Example Output

```
$ ./read_calib
camera: omnis5u3mt9v022:79fff4e28a810a00

sensor 0:
width = 752
height = 480
D = [[-0.44821,0.232306,0.000354156,0.00165202,-0.0631369]]
K = [[517.36,0,374.13];[0,520.01,200.115];[0,0,1]]
R = [[1,0.000791722,-9.2749e-05];[-0.000791778,1,-0.000598753];[9.22749e-05,0.
↪000598826,1]]
T = [[-0.383266,0.301836,-23.0616]]
sensor 1:
width = 752
height = 480
D = [[-0.453541,0.243334,0.00102948,0.000666533,-0.0687729]]
K = [[513.467,0,387.589];[0,515.589,196.805];[0,0,1]]
R = [[0.308474,-0.00268466,-0.951229];[0.00222121,0.999995,-0.00210198];[0.95123,-0.
↪00146447,0.308479]]
T = [[-2.04496,-0.510067,-25.3537]]
sensor 2:
width = 752
height = 480
D = [[-0.428696,0.186455,0.0010419,0.000630139,-0.0377135]]
K = [[512.636,0,376.636];[0,514.277,181.63];[0,0,1]]
```

```
R = [[-0.808822,0.00270545,-0.588047];[0.00754549,0.999955,-0.00577782];[0.588005,-0.
↪00911033,-0.808806]]
T = [[-1.0354,-4.9767,-24.6256]]
sensor 3:
width = 752
height = 480
D = [[-0.454364,0.240541,0.00100409,0.000646399,-0.0662772]]
K = [[517.499,0,387.135];[0,519.242,207.67];[0,0,1]]
R = [[-0.806489,-0.0231745,0.590795];[-0.0220254,0.999716,0.00914811];[-0.590839,-0.
↪00563462,-0.80677]]
T = [[-0.132979,-4.2827,-23.5639]]

sensor 4:
width = 752
height = 480
D = [[-0.454038,0.244789,0.000875864,0.00155121,-0.0699441]]
K = [[515.903,0,363.543];[0,518.519,208.217];[0,0,1]]
R = [[0.309849,-0.00773396,0.950754];[0.00604222,0.999963,0.0061651];[-0.950767,0.
↪00383442,0.309884]]
T = [[-0.557794,0.724112,-24.5413]]
sensor 5:
width = 752
height = 480
D = [[-0.438919,0.221365,0.000573236,-0.000172672,-0.0587718]]
K = [[517.88,0,383.076];[0,520.345,207.827];[0,0,1]]
R = [[0.999739,-0.0073721,-0.0216307];[0.00725101,0.999958,-0.00567105];[0.0216716,0.
↪00551272,0.99975]]
T = [[-0.1162,-122.594,-24.3255]]
sensor 6:
width = 752
height = 480
D = [[-0.4424,0.232244,0.00096465,0.00104117,-0.0662799]]
K = [[524.723,0,377.566];[0,527.138,244.883];[0,0,1]]
R = [[0.302794,-0.0104779,-0.952998];[0.00492329,0.999943,-0.0094298];[0.953043,-0.
↪00183659,0.302829]]
T = [[-5.93322,-123.025,-24.1984]]
sensor 7:
width = 752
height = 480
D = [[-0.453381,0.25848,-0.000589916,0.00102728,-0.091246]]
K = [[520.023,0,368.401];[0,522.825,194.952];[0,0,1]]
R = [[-0.813031,-0.00284479,-0.582214];[0.0176941,0.999405,-0.0295922];[0.581952,-0.
↪0343611,-0.812497]]
T = [[-2.08692,-126.91,-21.8086]]

sensor 8:
width = 752
height = 480
D = [[-0.455079,0.253845,0.000404952,-3.96015e-06,-0.0816325]]
K = [[516.193,0,354.8];[0,518.396,192.93];[0,0,1]]
R = [[-0.801857,-0.0407486,0.596125];[-0.0446589,0.998969,0.00821402];[-0.595845,-0.
↪0200358,-0.802849]]
T = [[2.38873,-126.946,-21.4451]]
sensor 9:
width = 752
height = 480
D = [[-0.443618,0.223433,0.00113362,7.36627e-05,-0.0574436]]
K = [[516.928,0,367.129];[0,519.509,196.242];[0,0,1]]
```

```
R = [[0.324493,-0.0146827,0.945774];[-0.000837974,0.999875,0.0158101];[-0.945888,-0.
↪00592281,0.32444]]
T = [[-0.959276,-121.773,-24.232]]
```

9.7 API Reference

9.7.1 Initialization

int occamInitialize();
Initialize the SDK. This function must be called before any other APIs.

Returns OCCAM_API_SUCCESS on success, otherwise OCCAM_API_NOT_INITIALIZED.

int occamShutdown();
Shut down the SDK. This function can optionally be called to free SDK allocated resources.

Returns OCCAM_API_SUCCESS

9.7.2 Memory Allocation

void* occamAlloc(
int size);
Allocate memory. This is a wrapper around malloc(), useful if the library is built with static CRT. Any memory that is returned from any module call that is expected to be freed by the user, must be allocated with this function and freed with #occamFree.

Returns the pointer to the allocated memory, or NULL if the allocation failed.

void occamFree(
void* ptr);
Free memory previously allocated with occamAlloc. This is a wrapper around free(), useful if the library is built with static CRT. Any memory that is returned from any module call that is expected to be freed by the user, must be allocated with #occamAlloc and freed with this call.

9.7.3 Device Enumeration

int occamEnumerateDeviceList(
int timeout_ms,
OccamDeviceList ret_device_list);**
Enumerate the set of devices present on the system. Uses libusb (all platforms), cyusb (Windows), libusbk (Windows) to enumerate the Occam devices supported by the SDK.

Parameters

- **timeout_ms** – the maximum amount of time this call should block while performing IO to enumerate devices.
- **ret_device_list** – the returned list of devices, on success. You must free this list using #occamFreeDeviceList.

Returns OCCAM_API_SUCCESS on success, and otherwise OCCAM_API_ERROR_ENUMERATING_DEVICES.

int occamFreeDeviceList(

```
OccamDeviceList* device_list);
```

Free a device list previous returned by a successful call to #occamEnumerateDeviceList.

Parameters

- **device_list** – the device list to free.

Returns always returns OCCAM_API_SUCCESS.

9.7.4 Opening and Closing Devices

```
int occamOpenDevice(
const char* cid,
OccamDevice** device);
```

Open a device.

Parameters

- **cid** – the unique device identifier, as given in the device list returned by #occamEnumerateDeviceList.
- **device** – the returned device, on success.

Returns OCCAM_API_SUCCESS if the device is successfully opened. OCCAM_API_INVALID_PARAMETER is returned if the unique device identifier is not valid

```
int occamCloseDevice(
OccamDevice* device);
```

Close a device.

Parameters

- **device** – the point to the open device, returned by occamOpenDevice.

Returns OCCAM_API_SUCCESS if successful.

```
int occamResetDevice(
OccamDevice* device);
```

Reset the device. This causes a soft reset on the hardware.

Parameters

- **device** – pointer to open device.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_WRITE_ERROR or OCCAM_API_READ_ERROR on error.

9.7.5 Device Parameters

```
int occamEnumerateParamList(
OccamDevice* device,
OccamParamList** ret_param_list);
```

Enumerate the set of parameters supported by a device. Use #occamFreeParamList to free the parameter list returned on success.

Parameters

- **device** – the open device to enumerate parameters of.
- **ret_param_list** – the returned parameter list.

Returns OCCAM_API_SUCCESS on success.

```
int occamFreeParamList(
OccamParamList* param_list);
```

Free memory associated with a parameter list returned by #occamEnumerateParamList.

```
int occamSetDeviceValuei(
OccamDevice* device,
OccamParam id,
int value);
```

Set an integer parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to set.
- **value** – the new value of the parameter.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamSetDeviceValuer(
OccamDevice* device,
OccamParam id,
double value);
```

Set an double parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to set.
- **value** – the new value of the parameter.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamSetDeviceValues(
OccamDevice* device,
OccamParam id,
const char* value);
```

Set an string parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to set.
- **value** – the new value of the parameter.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamSetDeviceValueiv(
OccamDevice* device,
OccamParam id,
const int* values,
```

```
int value_count);
```
Set a integer array parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to set.
- **values** – the new values of the parameter.
- **value_count** – the number of elements in the values array.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamSetDeviceValuerv(
OccamDevice* device,
OccamParam id,
const double* values,
int value_count);
```
Set a double array parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to set.
- **values** – the new values of the parameter.
- **value_count** – the number of elements in the values array.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_INVALID_COUNT if the size of the array is incorrect, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamSetDeviceValuesv(
OccamDevice* device,
OccamParam id,
char** values,
int value_count);
```
Set a string array parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to set.
- **values** – the new values of the parameter.
- **value_count** – the number of elements in the values array.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_INVALID_COUNT if the size of the array is incorrect, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamGetDeviceValuei(
OccamDevice* device,
OccamParam id,
```

int* value);

Get an integer parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **value** – pointer to value to be assigned.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_INVALID_COUNT if the size of the array is incorrect, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

int occamGetDeviceValuer(
OccamDevice* device,
OccamParam id,
double* value);

Get an integer parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **value** – pointer to value to be assigned.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

int occamGetDeviceValues(
OccamDevice* device,
OccamParam id,
char value);**

Get a double parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **value** – pointer to value to be assigned.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

int occamGetDeviceValuep(
OccamDevice* device,
OccamParam id,
void value);**

Get an opaque pointer to internal parameter. This is typically used to get module handles of internal components.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **value** – pointer to value to be assigned.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamGetDeviceValueiv(
OccamDevice* device,
OccamParam id,
int* values,
int value_count);
```

Get a string parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **values** – pointer to values to be assigned.
- **value_count** – the number of elements in the values array.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamGetDeviceValuerv(
OccamDevice* device,
OccamParam id,
double* values,
int value_count);
```

Get an integer array parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **values** – pointer to values to be assigned.
- **value_count** – the number of elements in the values array.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_INVALID_COUNT if the size of the array is incorrect, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamGetDeviceValuesv(
OccamDevice* device,
OccamParam id,
char** values,
int value_count);
```

Get a string array parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **values** – pointer to values to be assigned.
- **value_count** – the number of elements in the values array.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_INVALID_COUNT if the size of the array is incorrect, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamGetDeviceValuepv(
OccamDevice* device,
OccamParam id,
void** values,
int value_count);
```

Get a pointer array parameter.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **values** – pointer to values to be assigned.
- **value_count** – the number of elements in the values array.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device, OCCAM_API_INVALID_COUNT if the size of the array is incorrect, OCCAM_API_GENERIC_ERROR or a more specific error if there is a failure communicating with the hardware.

```
int occamGetDeviceValueCount(
OccamDevice* device,
OccamParam id,
int* value_count);
```

Get the number of elements for array parameters.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to get.
- **value_count** – pointer to integer that contains the number of elements on return.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device.

```
int occamResetDeviceValue(
OccamDevice* device,
OccamParam id);
```

Reset the given parameter to its factory default.

Parameters

- **device** – the device to set the parameter on.
- **id** – the parameter to reset.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_INVALID_PARAMETER if the parameter is not supported by the device.

9.7.6 Reading Device Output

```
int occamDeviceReadData(
OccamDevice* device,
```

```
int req_count,
const OccamDataName* req,
OccamDataType* ret_types,
void** ret_data,
int block);
```

Read a single frame of data. Reads a single frame of data in one or more types of returned forms. The minimal set of operations to satisfy the request will be formed. If a supported data is not requested then it will not be computed if possible.

Parameters

- **device** – pointer to open device.
- **req_count** – the number of outputs requested.
- **req** – array of data names that are requested.
- **ret_types** – the returned data types of the data returned. May be null.
- **ret_data** – the returned data.
- **block** – whether the function should block. If this is false and no data is available, then OCCAM_API_DATA_NOT_AVAILABLE is returned. Otherwise the calling code will block until data becomes ready or an error occurs (e.g., device goes away).

Returns OCCAM_API_SUCCESS on success, OCCAM_API_UNSUPPORTED_DATA if data is requested that is not supported by the device. Other errors may be returned by the driver in the case of hardware failure.

```
int occamDeviceAvailableData(
OccamDevice* device,
int* req_count,
OccamDataName** req,
OccamDataType** types);
```

Query the driver for what data is available. The available data may depend on the configuration of the device according to device values. Note that on successful return, the values req and types must be freed via an #occamFree call.

Parameters

- **device** – pointer to open device.
- **req_count** – pointer to returned value indicating number of elements in returned req and types arrays.
- **req** – the set of data names returned.
- **req** – the set of data types returned.

Returns OCCAM_API_SUCCESS on success.

9.7.7 Device Registers

```
int occamWriteRegister(
OccamDevice* device,
uint32_t addr,
uint32_t value);
```

Write a register on the device.

Parameters

- **device** – pointer to open device.

- **addr** – the register index.
- **value** – the register value.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_WRITE_ERROR on error.

```
int occamReadRegister(
OccamDevice* device,
uint32_t addr,
uint32_t* value);
```

Write a register on the device.

Parameters

- **device** – pointer to open device.
- **addr** – the register index.
- **value** – pointer to returned register value.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_READ_ERROR or OCCAM_API_WRITE_ERROR on error.

9.7.8 Device Storage

```
int occamWriteStorage(
OccamDevice* device,
uint32_t target,
uint32_t addr,
uint32_t len,
const uint8_t* data);
```

Write to device storage.

Parameters

- **device** – pointer to open device.
- **target** – the device request type. These are device specific values.
- **addr** – the data address. These are device specific values.
- **len** – number of bytes in supplied data array.
- **data** – array of bytes supplied for the request.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_WRITE_ERROR or OCCAM_API_READ_ERROR on error.

```
int occamReadStorage(
OccamDevice* device,
uint32_t target,
uint32_t addr,
uint32_t len,
uint8_t* data);
```

Read from device storage.

Parameters

- **device** – pointer to open device.
- **target** – the device request type. These are device specific values.
- **addr** – the data address. These are device specific values.

- **len** – number of bytes in supplied data array.
- **data** – array of bytes supplied for the request.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_WRITE_ERROR or OCCAM_API_READ_ERROR on error.

```
int occamSaveSettings(
OccamDevice* device);
```

Store settings to non-volatile memory on the device.

Parameters

- **device** – pointer to open device.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_WRITE_ERROR or OCCAM_API_READ_ERROR on error.

9.7.9 Images

```
int occamFreeImage(
OccamImage* image);
```

Frees the given image. This reduces the reference count on the image and actually frees memory when that reaches zero.

Parameters

- **image** – the image to free.

Returns OCCAM_API_SUCCESS on success.

```
int occamSubImage(
OccamImage* image,
OccamImage** new_image,
int x,
int y,
int width,
int height);
```

Returns a sub-image of a given image. Note that this allocates a new OccamImage structure but points to the original image memory.

Parameters

- **image** – the image to use.
- **new_image** – a pointer to the returned OccamImage.
- **x** – the left of the sub-image in pixels.
- **y** – the top of the sub-image in pixels.
- **width** – the width of the sub-image in pixels.
- **height** – the height of the sub-image in pixels.

Returns OCCAM_API_SUCCESS on success, or OCCAM_API_INVALID_PARAMETER.

```
int occamCopyImage(
const OccamImage* image,
OccamImage** new_image,
int deep_copy);
```

Copies the given image. If deep_copy is non-zero, the actual image data will be copied. Otherwise the returned OccamImage points to the same memory and increases the reference count on the structure.

Parameters

- **image** – the image to copy.
- **new_image** – a pointer to the returned OccamImage.
- **deep_copy** – whether to actually copy image data or just increase reference count.

Returns OCCAM_API_SUCCESS on success, or OCCAM_API_INVALID_PARAMETER.

```
int occamImageFormatPlanes(
OccamImageFormat format,
int* planes);
```

Returns the number of planes implied by the given image format.

Parameters

- **format** – the image format.
- **planes** – pointer to integer with returned number of planes.

Returns OCCAM_API_SUCCESS on success, or OCCAM_API_INVALID_PARAMETER if format not known.

```
int occamImageFormatBytesPerPixel(
OccamImageFormat format,
int* bpp);
```

Returns the number of bytes per pixel implied by the given image format.

Parameters

- **format** – the image format.
- **bpp** – pointer to integer with returned number of bytes per pixels.

Returns OCCAM_API_SUCCESS on success, or OCCAM_API_INVALID_PARAMETER if format not known.

9.7.10 Point Clouds

```
int occamFreePointCloud(
OccamPointCloud* point_cloud);
```

Frees the given point cloud. This reduces the reference count on the image and actually frees memory when that reaches zero.

Parameters

- **image** – the point cloud to free.

Returns OCCAM_API_SUCCESS on success.

```
int occamCopyPointCloud(
const OccamPointCloud* point_cloud,
OccamPointCloud** new_point_cloud,
int deep_copy);
```

Copies the given point cloud. If deep_copy is non-zero, the actual data will be copied. Otherwise the returned OccamPointCloud points to the same memory and increases the reference count on the structure.

Parameters

- **point_cloud** – the point cloud to copy.
- **new_point_cloud** – a pointer to the returned OccamPointCloud.

- **deep_copy** – whether to actually copy data or just increase reference count.

Returns OCCAM_API_SUCCESS on success, or OCCAM_API_INVALID_PARAMETER.

9.7.11 Markers

```
int occamFreeMarkers(
OccamMarkers* markers);
```

Frees the given markers. This reduces the reference count on the markers and actually frees memory when that reaches zero.

Parameters

- **markers** – the markers to free.

Returns OCCAM_API_SUCCESS on success.

```
int occamCopyMarkers(
const OccamMarkers* markers,
OccamMarkers** new_markers,
int deep_copy);
```

Copies the given markers. If deep_copy is non-zero, the actual marker data will be copied. Otherwise the returned OccamMarkers points to the same memory and increases the reference count on the structure.

Parameters

- **markers** – the markers to copy.
- **new_markers** – a pointer to the returned OccamMarkers.
- **deep_copy** – whether to actually copy marker data or just increase reference count.

Returns OCCAM_API_SUCCESS on success, or OCCAM_API_INVALID_PARAMETER.

```
int occamGetMarkerField(
const OccamMarkers* markers,
int i,
OccamMarkerFieldName name,
OccamMarkerFieldType type,
void* value);
```

Read a types marker field. The data pointer must point to pre-allocated space or reside on the stack and be of sufficient size corresponding to specified type. The resulting field is converted into the specific data type as required.

Parameters

- **markers** – the marker to read from.
- **i** – the index of the marker.
- **name** – the name of the field within the marker.
- **type** – the type of the field.
- **value** – pointer that will contain the resulting value.

Returns OCCAM_API_SUCCESS on success, OCCAM_API_FIELD_NOT_FOUND if field is not found, OCCAM_API_INVALID_TYPE if implied type cast is not possible.

CHAPTER

TEN

CALIBRATION

10.1 Factory Calibration

All cameras come factory calibrated, and normally should not need to be recalibrated.

The calibration data is stored in the camera's non-volatile EEPROM memory. It can be programmatically read using the SDK API and the read_calib SDK example.

Over time and temperature variances the camera may become uncalibrated, and in that case the following procedure should be followed to re-calibrate the camera.

10.2 Calibrating Omni 60 and Omni Stereo

The procedure below assumes that you already have your camera up and running in Windows and working with the Indigo Tools graphical program that is part of the SDK.

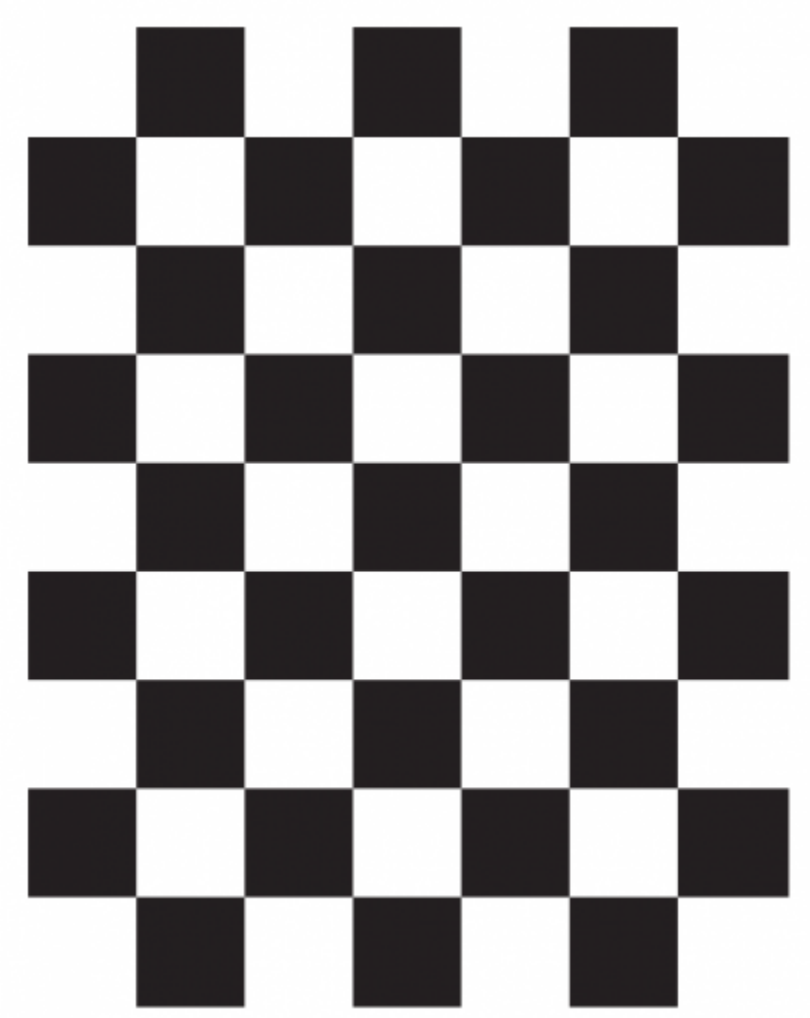
By calibration we mean the internal and external parameters of the five sensors in the camera. – The internal parameters consist of the standard OpenCV model. That is, fx, fy, cx, cy and 5 distortion numbers $k1$, $k2$, $k3$, $k4$, $k5$ (the standard OpenCV radial polynomial model) describing the transformation from camera to image coordinates, and then to final distorted image coordinates. – The external parameters consist of a 3×3 rotation matrix and 3×1 pre-rotated camera translation. These plug into the standard pinhole camera model $x = PX = [R, T]X$ that OpenCV uses. The center of this coordinate frame is the center of the ring of sensors inside the enclosure (roughly in the center of the enclosure).

The Indigo Tools software (Windows only) is required to run the calibration tool. The tool collects images of checkerboards, solves for the various calibration parameters, and then writes them to the camera EEPROM memory. The Windows tool is only required to solve for the parameters– once they are found you can use them from Linux or any other libusb-compatible platform by simply reading them from the camera via the SDK.

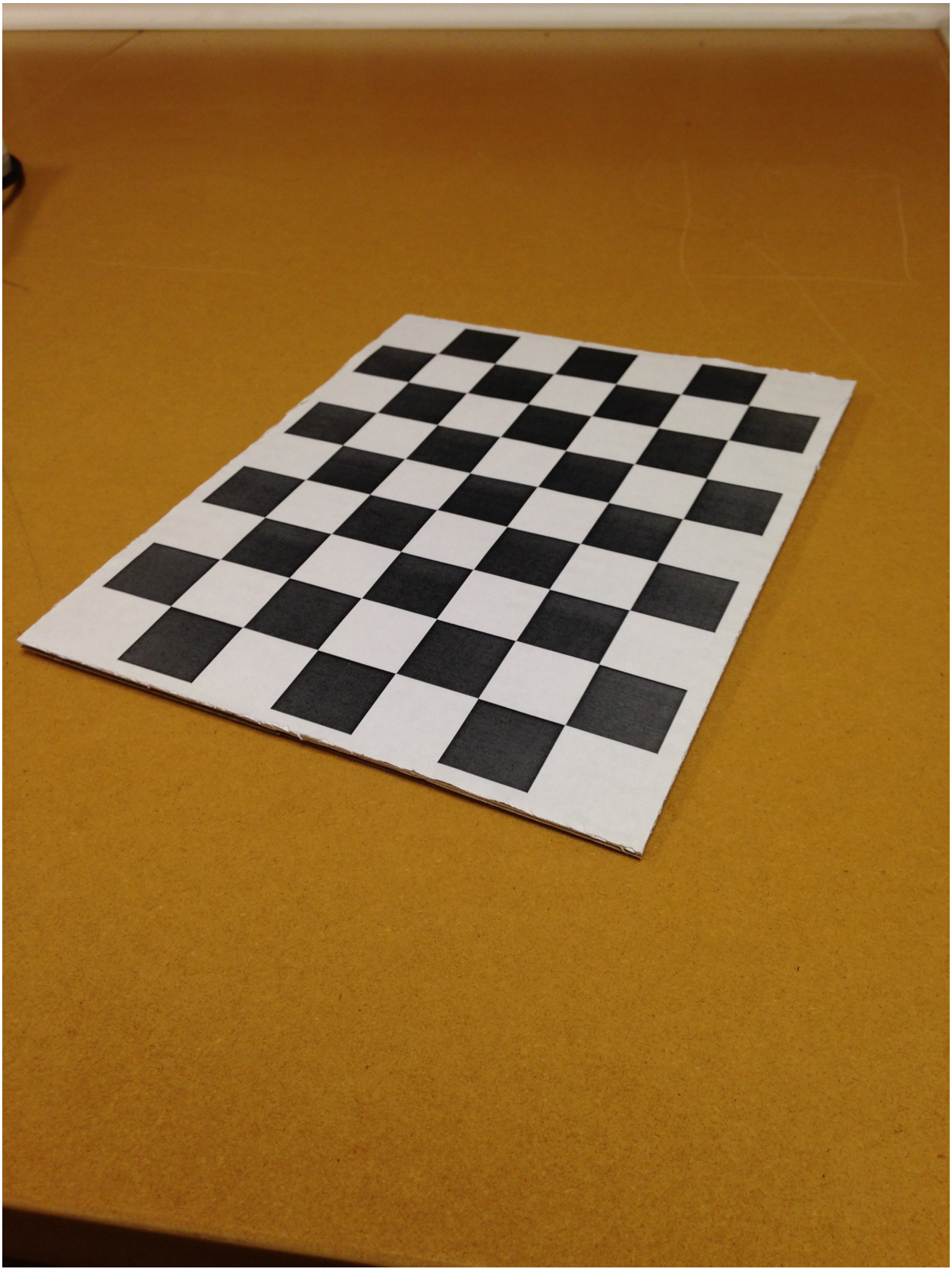
Also, once a camera is calibrated you can use the cylindrical stitching mode provided by the SDK to provide a seamless sensor image from the five raw sensor feeds.

- Step 1: Download and print checkerboard

First, you need to print the checkerboard pattern which can be downloaded from the occamvisiongroup.com web site, or was included in the offline media included with the camera. It looks like this:

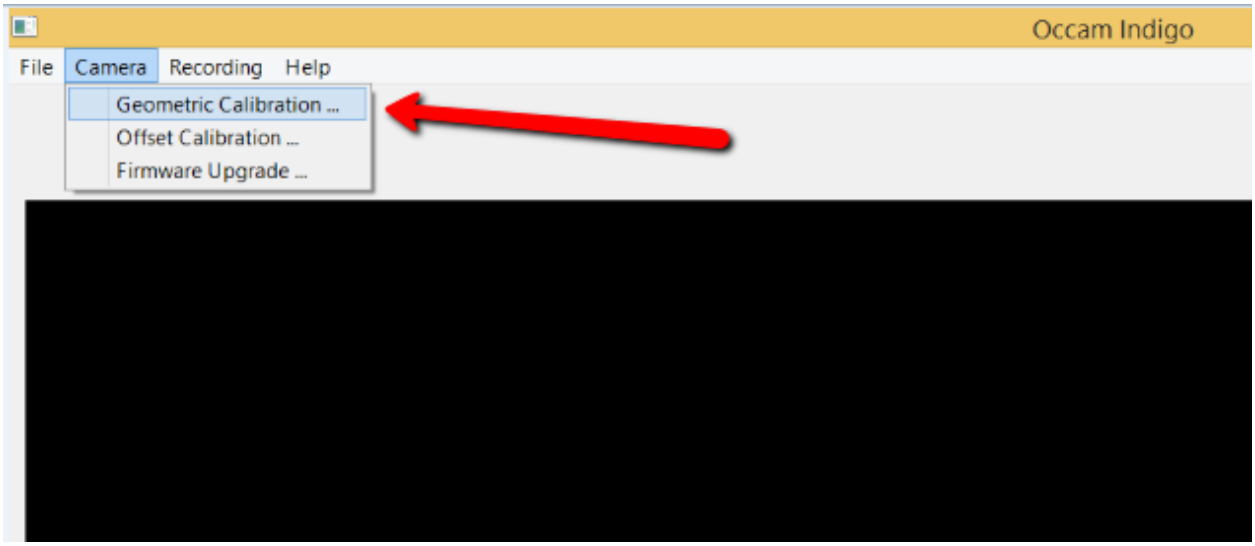


You should print it and glue it to a piece of posterboard or other hard surface. It is important that the sheet is rigid and flat when using it for calibration. You don't need to make this overly complicated– something as simple as applying it to a piece of cardboard with double sided tape usually gives good result. You can also have the board machined or commercially printed if accuracy is very important. Here's a simple cardboard mounting:



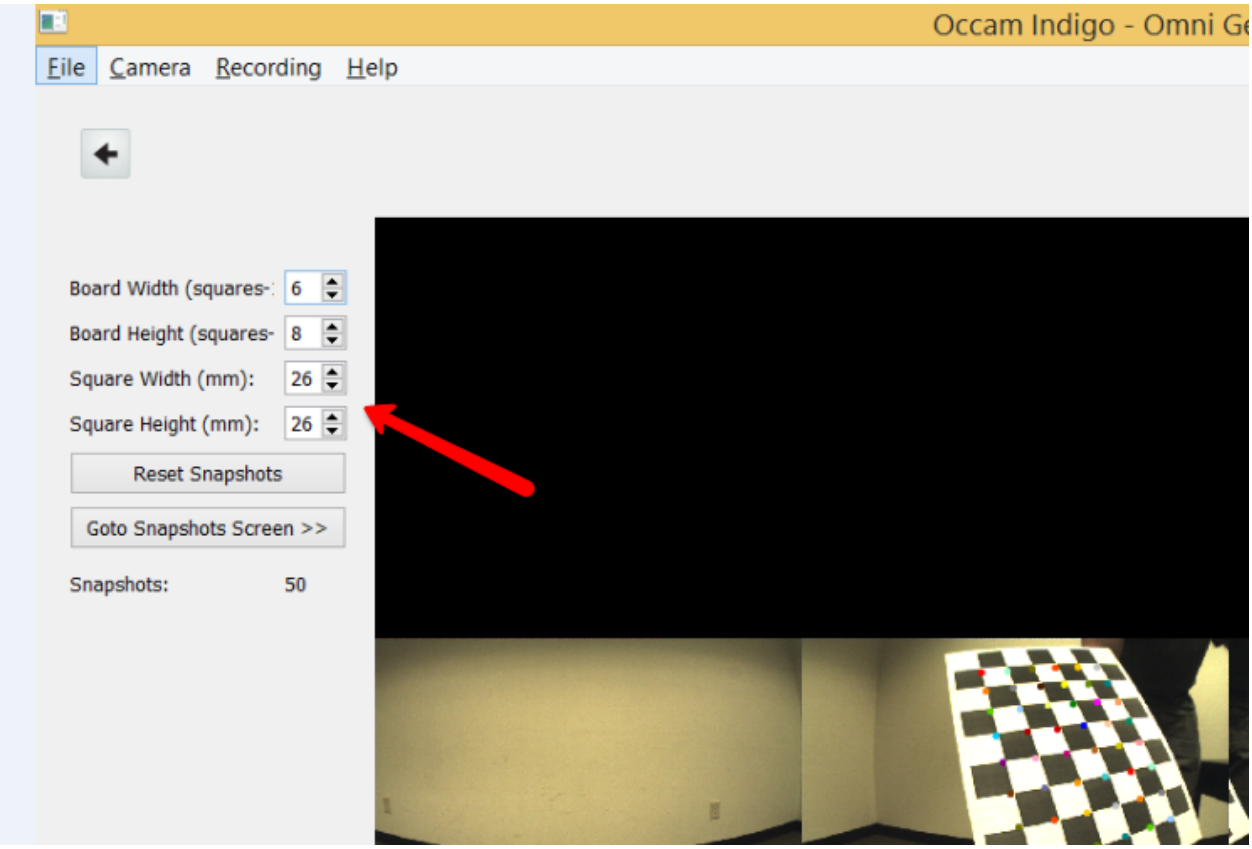
- Step 2: Open Calibration Tool

Open the Indigo Tools software, and open the Geometric Calibration tool.



- Step 3: Set checkerboard parameters

Enter the width and height of the black squares into the UI:

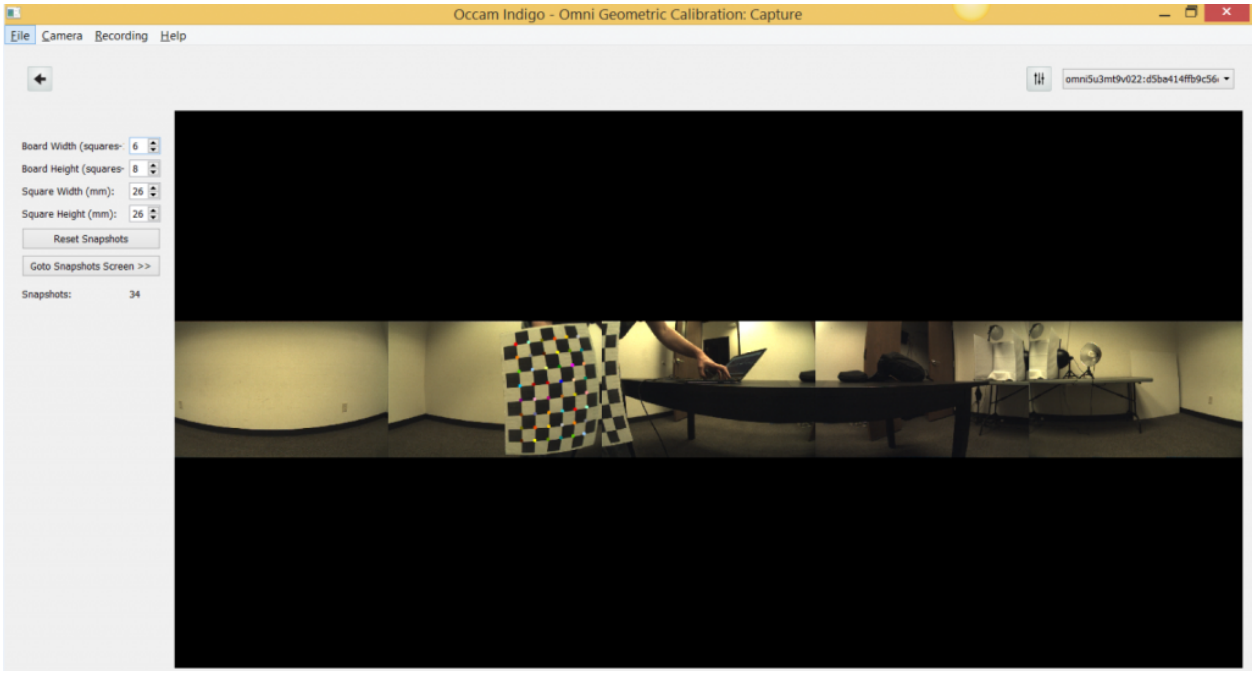


The numbers will depend on how you printed it. Here the squares are measured to be 26mm so that is entered into the tool.

- Step 4: Capture individual sensor snapshots

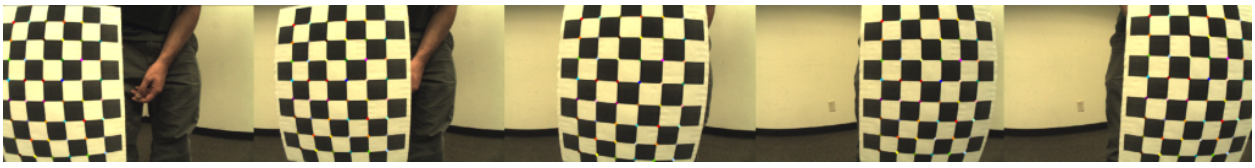
Hold the checkerboard still in front of a sensor until colored dots appear over the checkerboard as in the following

picture:

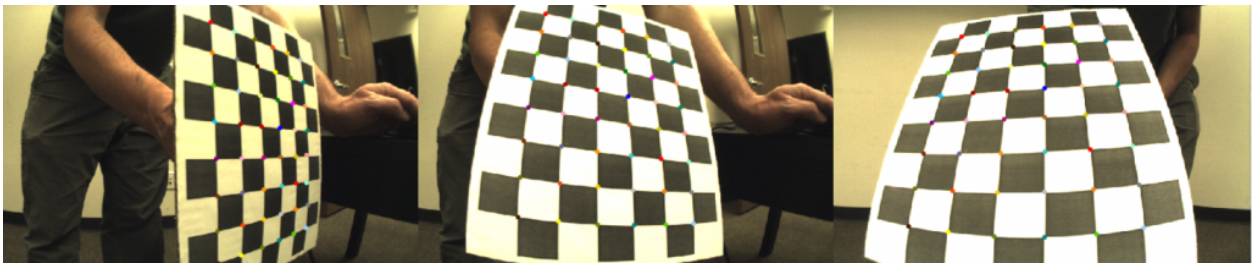


You will want to capture about 8 – 15 snapshots for each sensor. The position of the checkerboard and the types of orientations you capture are very important. For good results, you should usually capture two types of positions:

Sweep across the entire image as close to the camera as you can, so that the checkerboard is as close to the left edge for one frame, and then after a few frames as close to the right edge as possible. Here is an example:

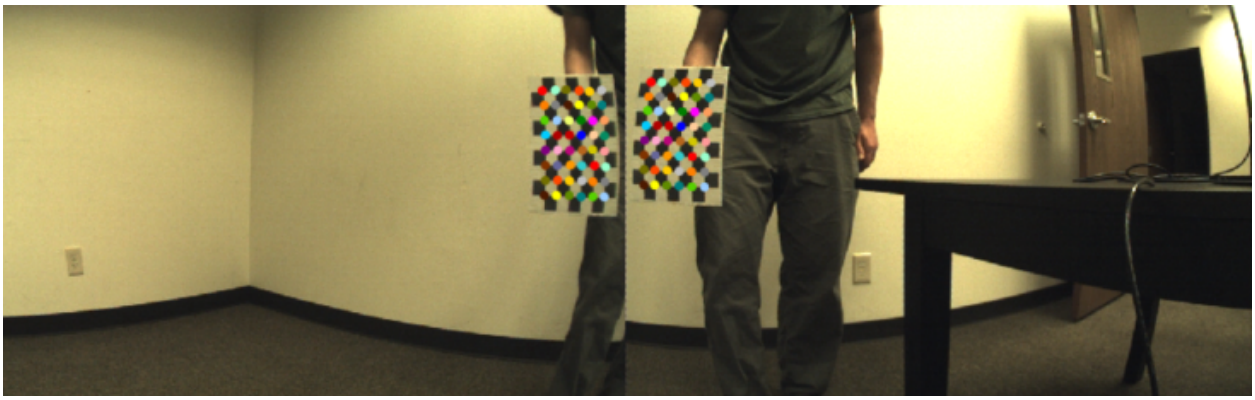


Then, capture several frames where the board is rotated with respect to the x and y axis of the image. Here's an example:



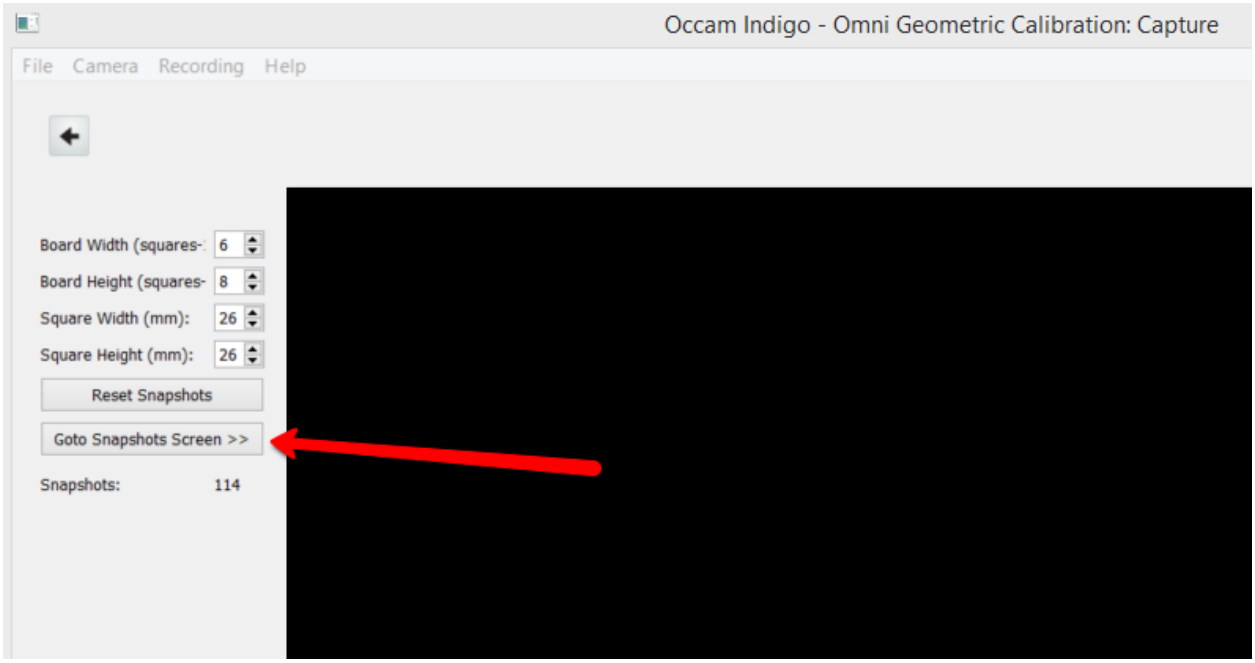
- Step 5: Capture inter-sensor snapshots

For each pair of adjacent sensors, capture a few snapshots that are seen by both of the sensors. Here's an example:



- Step 6: Select snapshots

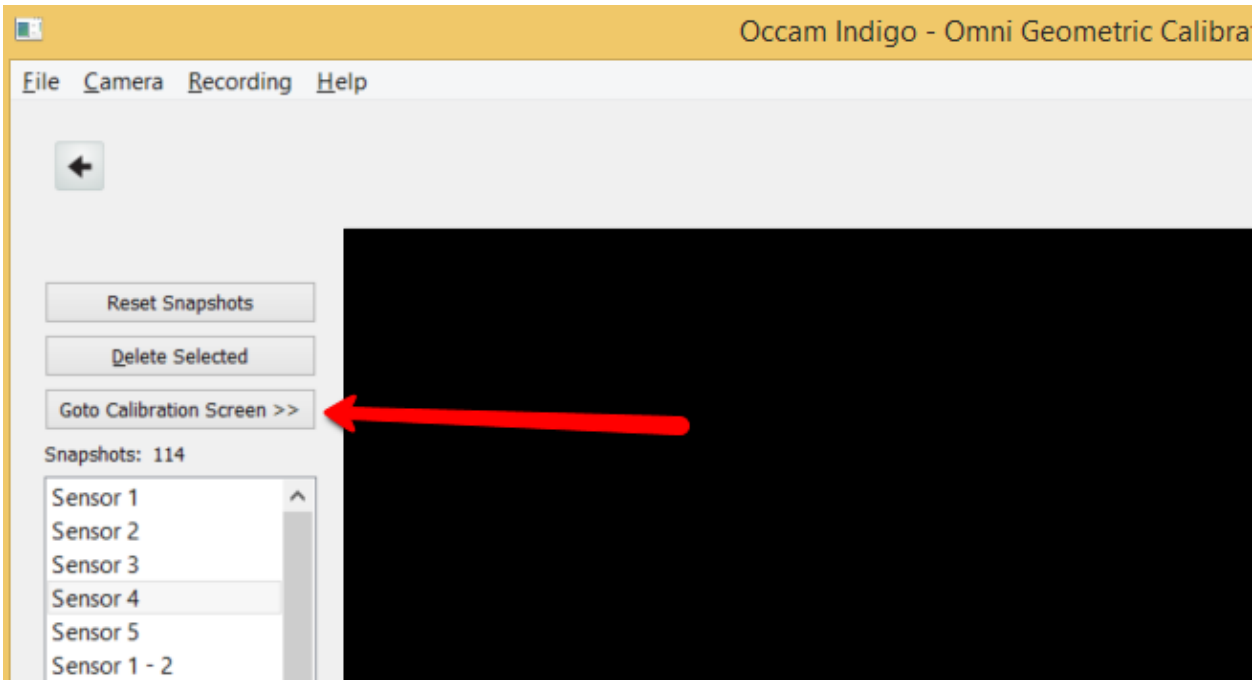
Once you capture the snapshots you need, you must select which ones get used by the calibration solver.



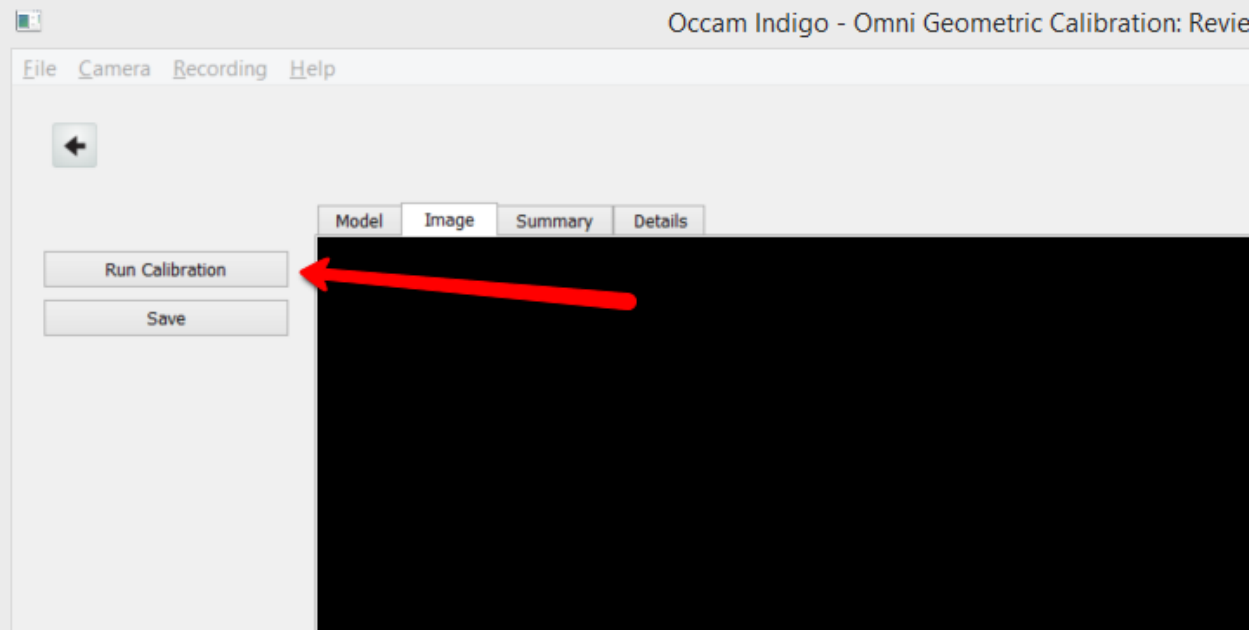
That will open the snapshots screen that shows two lists for each sensor and each pair of adjacent sensors. The top list contains all the snapshots captured except the ones marked for use, and the bottom list contains the ones marked for use. The idea here is that you double click each of the snapshots you want the solver to use to move them to the bottom list. Each individual sensor should get 8 – 15 snapshots, and each pair of sensors should get 1-3 snapshots around the center of the image. If some snapshots had bad data (motion blur, noisy points, etc) you can delete those snapshots or just not use them.

- Step 6: Run the calibration solver

Once you have selected the appropriate number of good snapshots for each list, click the Goto Calibration Screen button to go to the calibration solver screen.

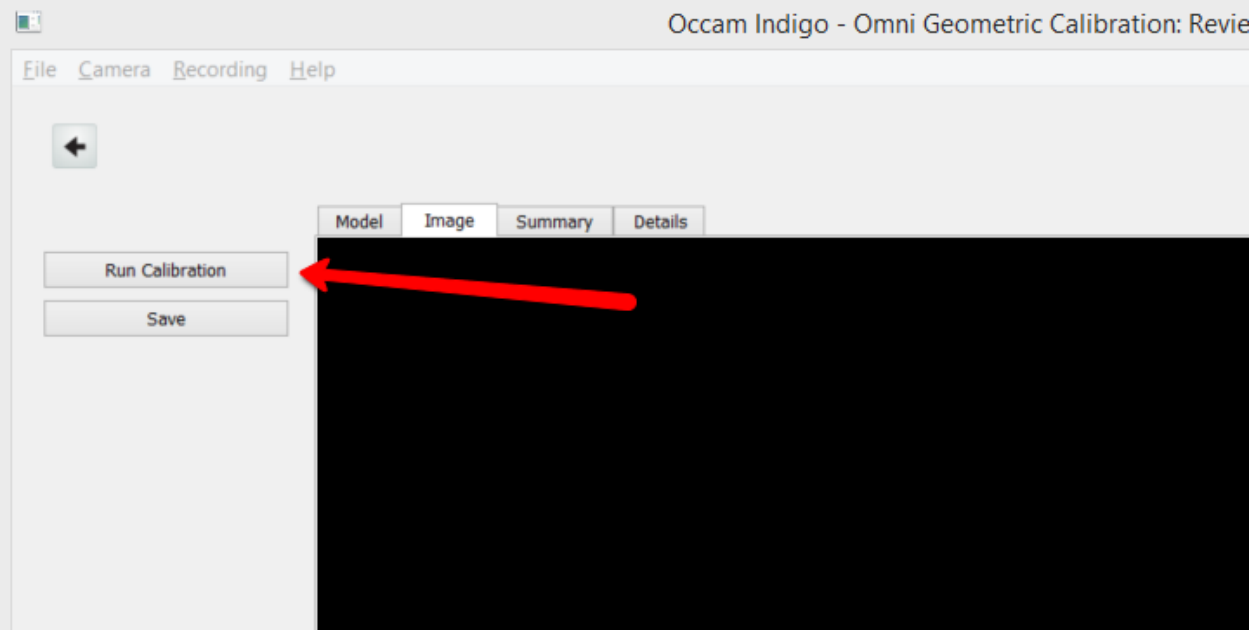


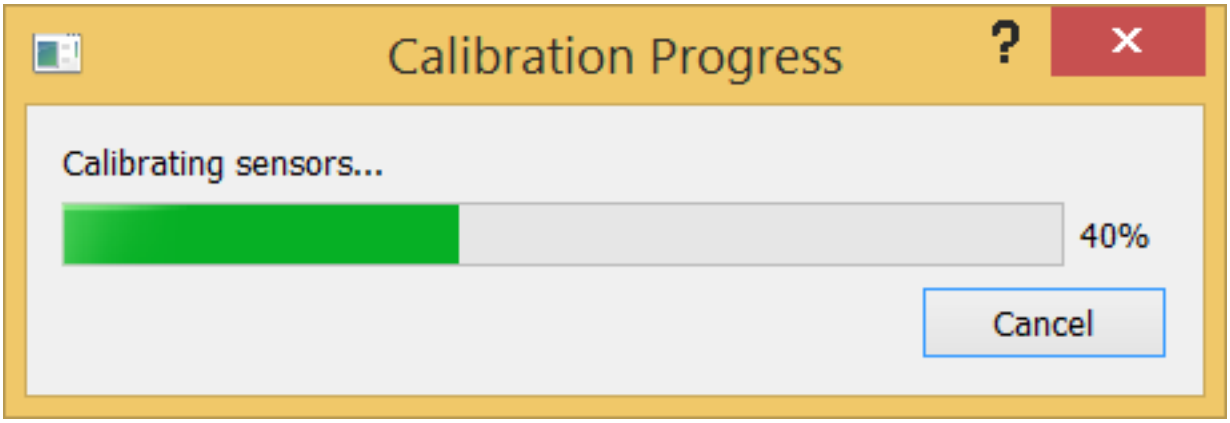
Then click the "Run Calibration" button to start the solve process.

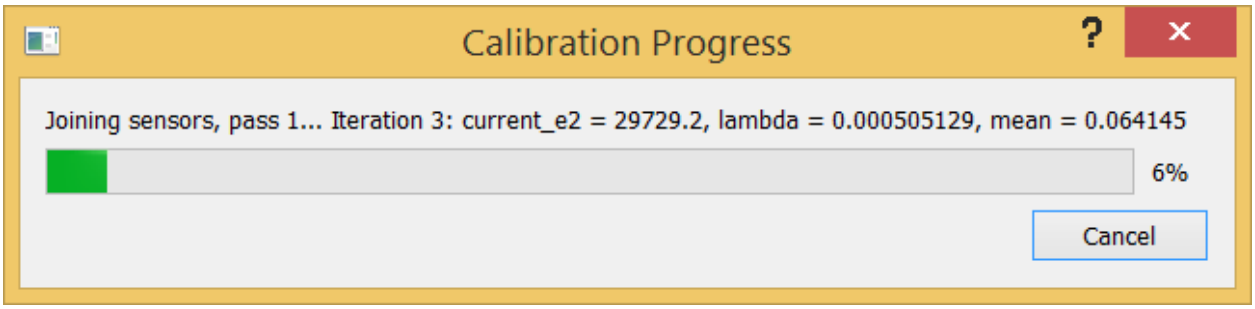


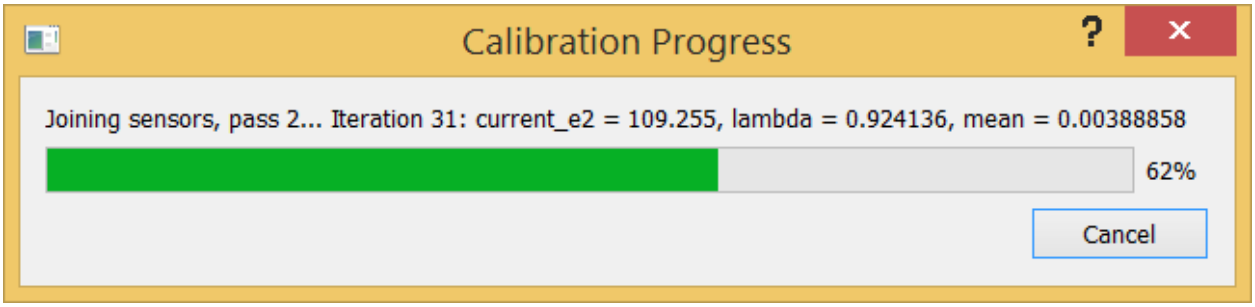
This will take a little while. No more than 1 minute on typical PC, often around 30 seconds. The time it takes also depends on the number of snapshots you selected to use.

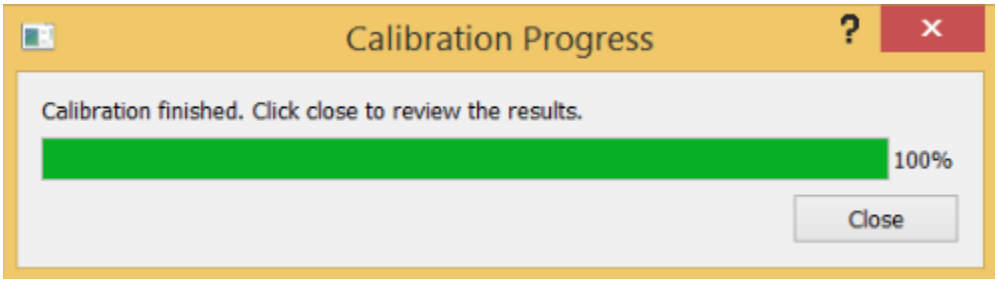
During the processing, some status information will be displayed indicating progress:







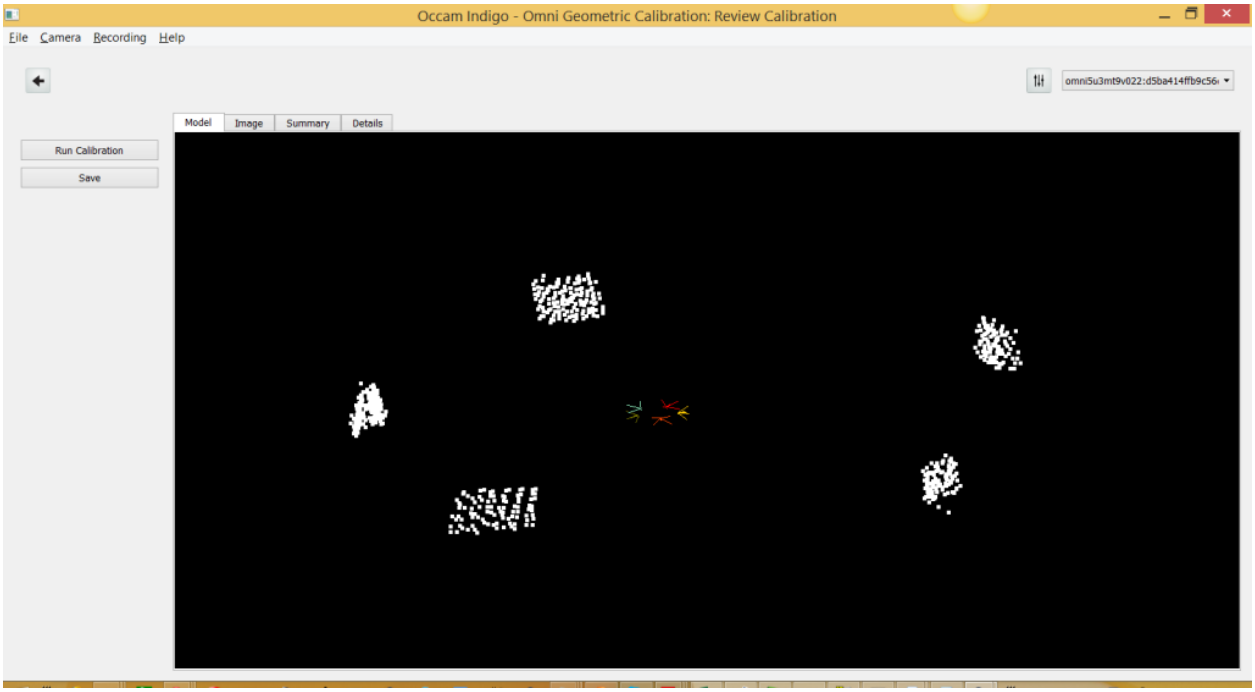




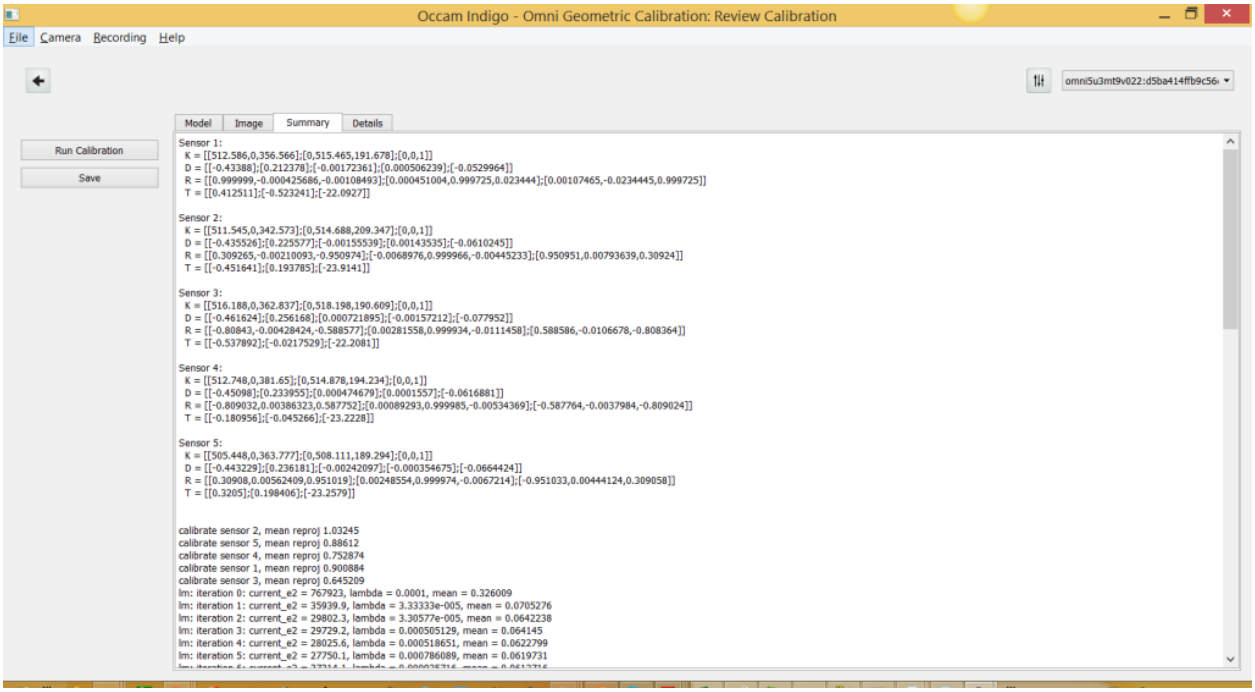
- Step 7: Inspect results

Click the Close button on the progress dialog, after which you can select the various tabs to get more information on the calibration result.

The model tab gives a rendering of the 3D model of the camera, showing the sensors within the camera coordinate frame as well as a point cloud of the inter-sensor checkerboard points. The result should look something like this:



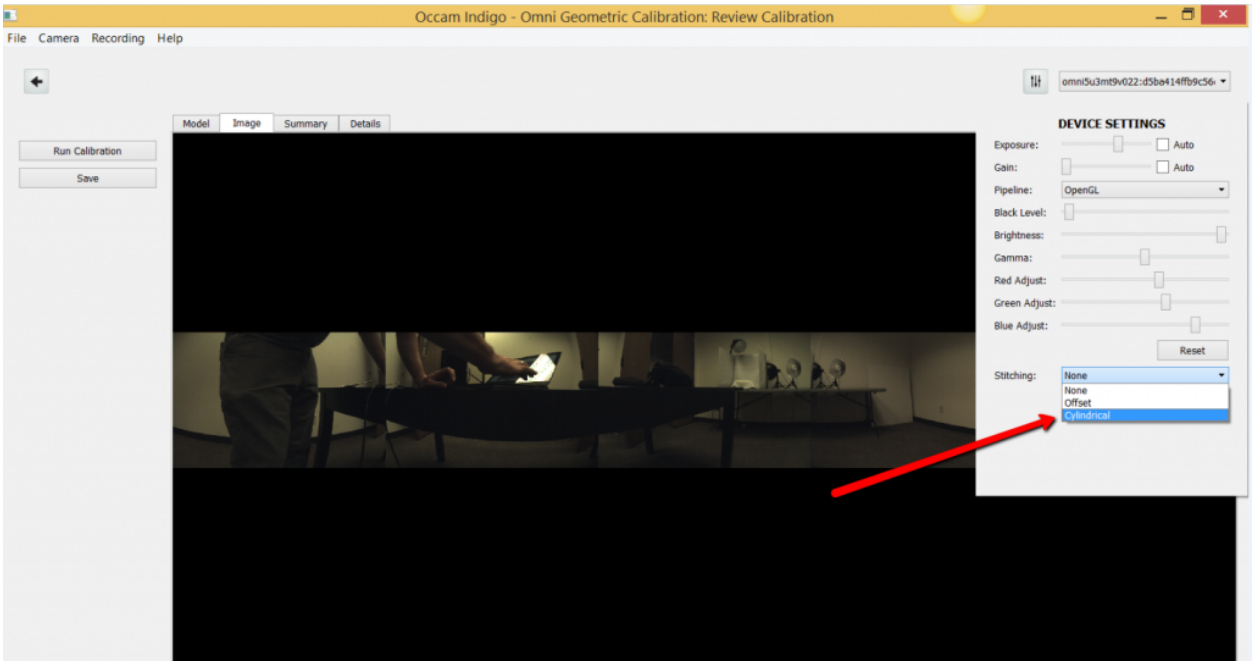
The summary tab gives a rundown of the numerical information. The calibration parameters themselves (the D, K, R, T matrices for each sensor) as well as error measures for various parts of the calibration process. The interesting numbers are the mean and median reprojection errors for individual sensor calibration and for the inter-sensor calibration. There are also ten-percentile reprojection errors given for the final reconstruction. If you have selected a good set of snapshots, typical results are less than 0.5 pixels with a distribution such as $[[0.000847245, 0.0341882, 0.0626213, 0.0929949, 0.122866, 0.153567, 0.196825, 0.243231, 0.289278, 0.363943]]$.



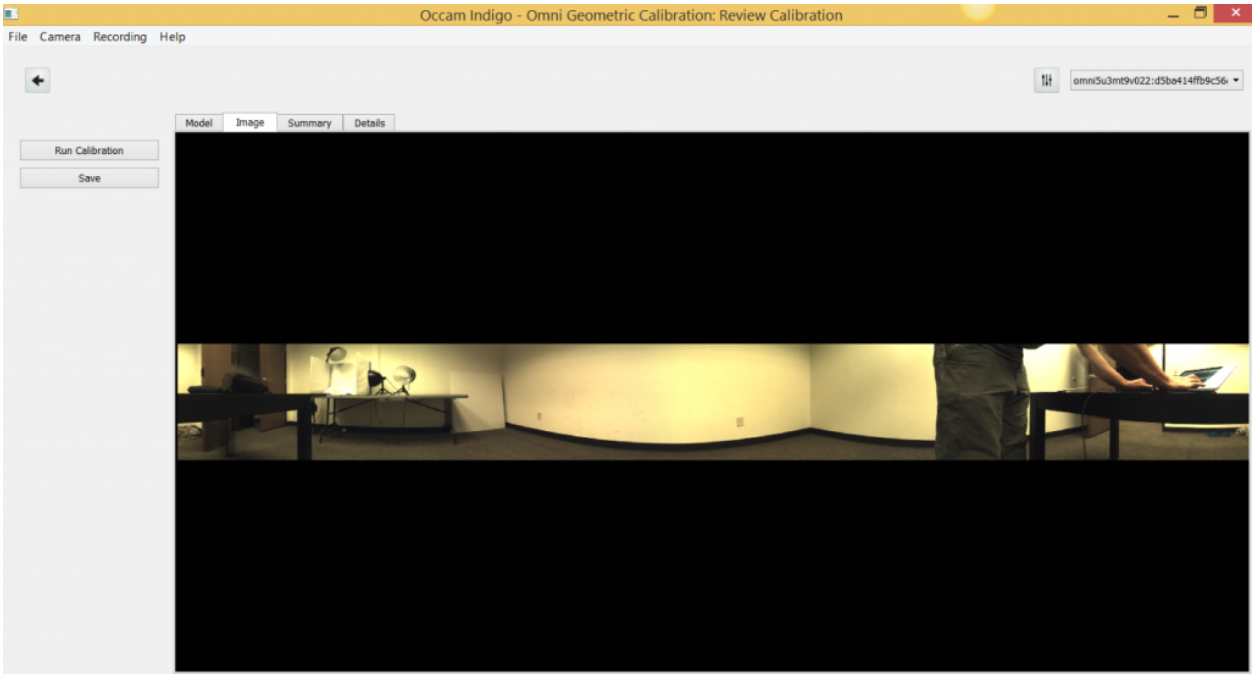
Typically the least accurate part of the calibration will be the lens distortion parameters D. Those are very sensitive to the snapshots that you chose for the individual sensors. You should re-run the calibration until you can achieve close to half-pixel reprojection errors or better. Also, if the distortion parameters are not correct, the final reprojection errors

will often be better or worse depending on how close the inter-sensor snapshots are to the center of the sensor they are since there will be the least distortion there.

The image tab shows the cylindrically stitched image. This method of stitching forms a single image by projecting a cylinder into each of the sensors. Click the camera settings button in the upper left to adjust the camera settings and select the Cylindrical stitching mode.



Now the image will be a single stitched image instead of five individual images, for example:

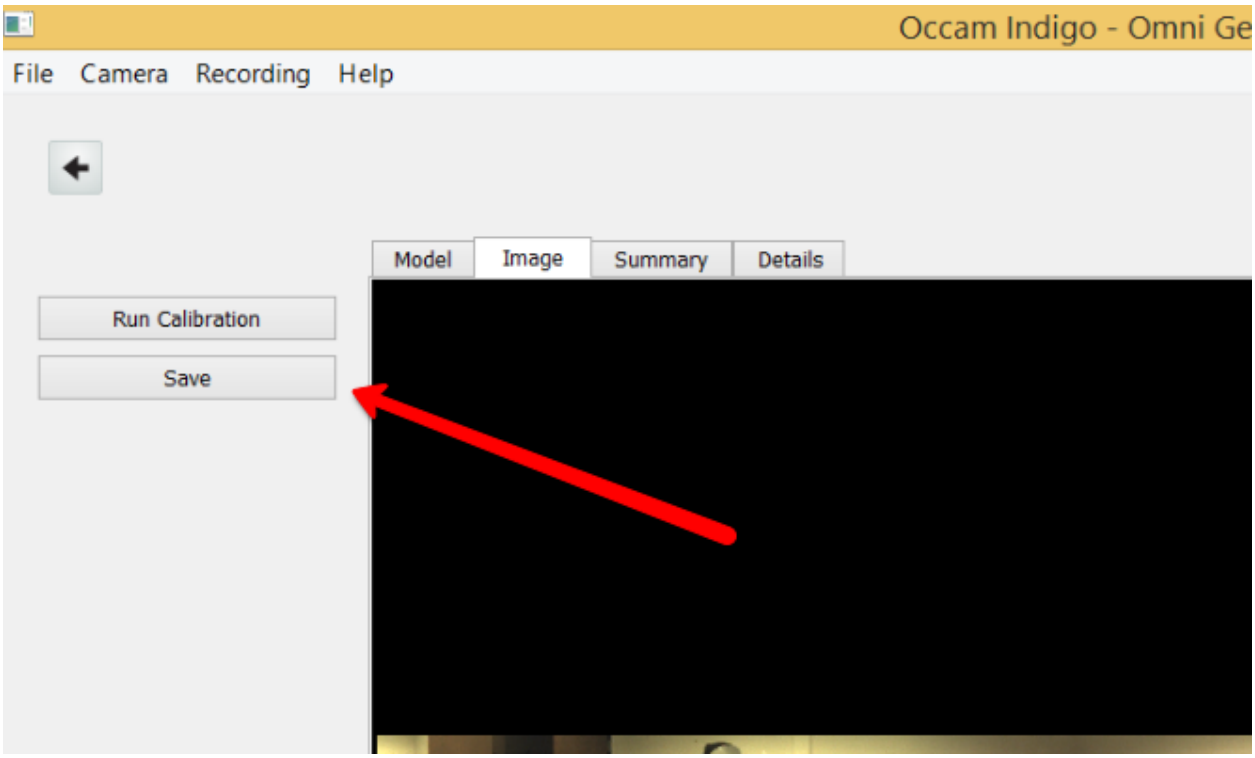


- Step 8: Save results to camera memory

If you are not satisfied with the calibration result, you must return to the snapshots screens and either select different snapshots, or adjust which snapshots to use for the calibration procedure.

If the calibration result looks good, click the Save button to store the data into the camera's non-volatile EEPROM memory.

You can also click the Save button in the camera settings pop-up where you selected the stitching mode (along with exposure, gain, etc), and these will come up as the default settings when you use the camera from the SDK in your own software or from ROS.



CHAPTER

ELEVEN

SYNCHRONIZATION

Omni 60 cameras are synchronized internally such that all five sensors begin exposing at the same time, and stop exposing at the same time.

Omni Stereo cameras are comprised of two Omni 60 cameras. Each of these Omni 60 cameras are themselves internally sychronized, and are externally synchronized with each other. The Omni Stereo synchronization happens over the USB connection to the host, using the Isochronous Timestamp Packets (ITP) feature of USB3. The actual timing enforcement does not depend on the host software, only on the USB3 controller hardware in the host. It does require that both the top half and bottom half of the camera are plugged into the same USB3 controller on the host.

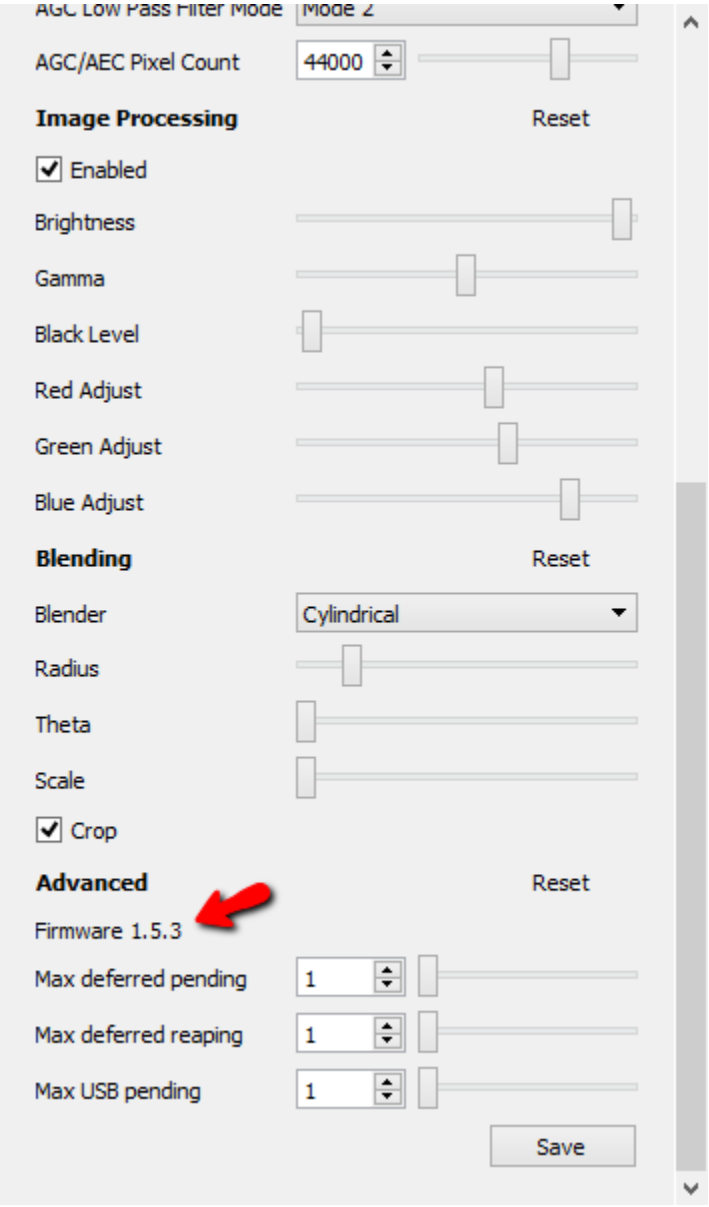
11.1 Frame Timecode

The `OccamImage` field `time_ns` gives the nanosecond time of the frame, and `index` gives a monotonically increasing index of the frame.

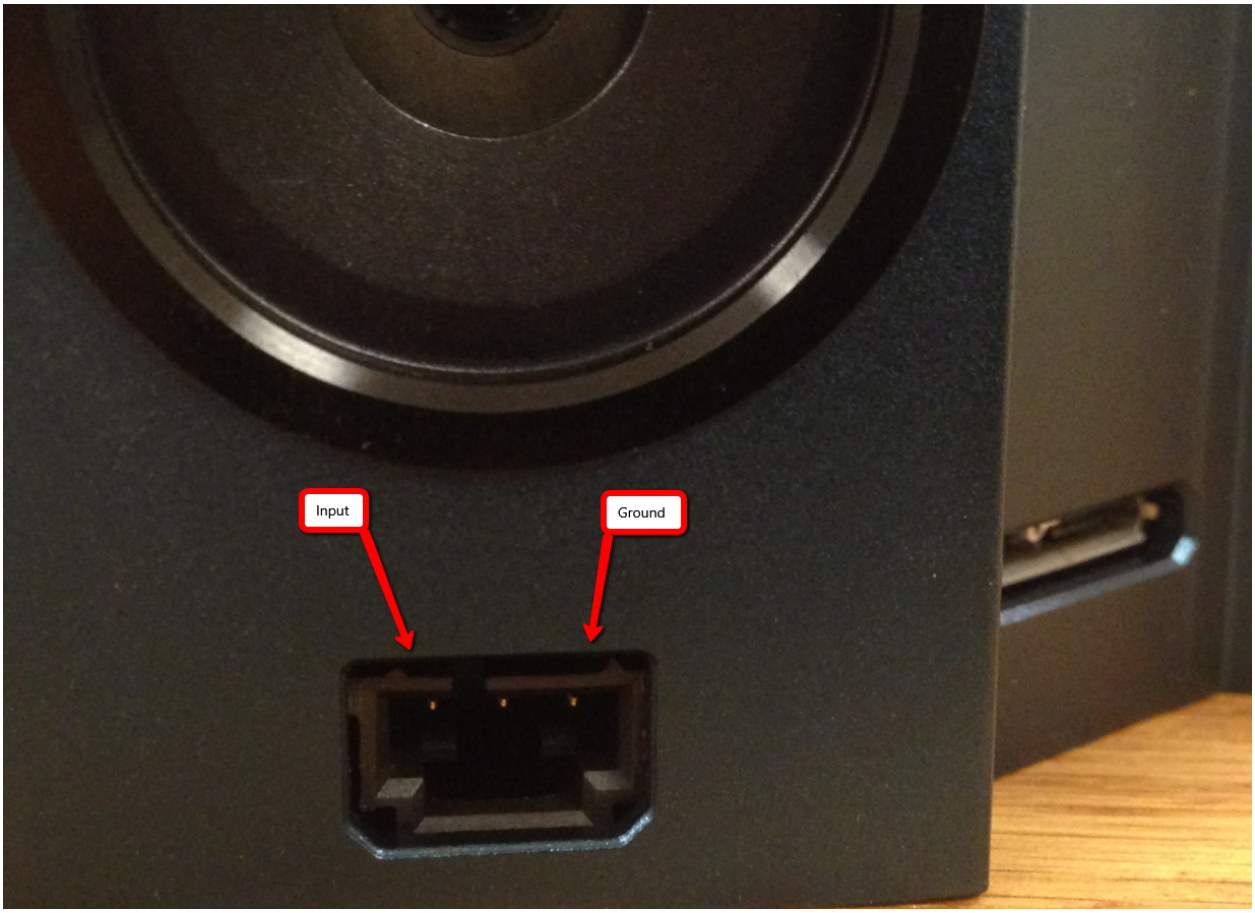
11.2 Omni 60 Trigger Input

The Omni 60 camera can be driven by externally trigger signal. To do this, you must put the sensors into "snapshot mode" while driving a trigger pulse train on the GPIO input port of the camera. You may trigger the camera at up to 30 FPS using this method.

The camera must be upgraded to at least firmware revision 1.5.3. You can see the version of the firmware you have by clicking the settings drop-down and scrolling to the bottom:

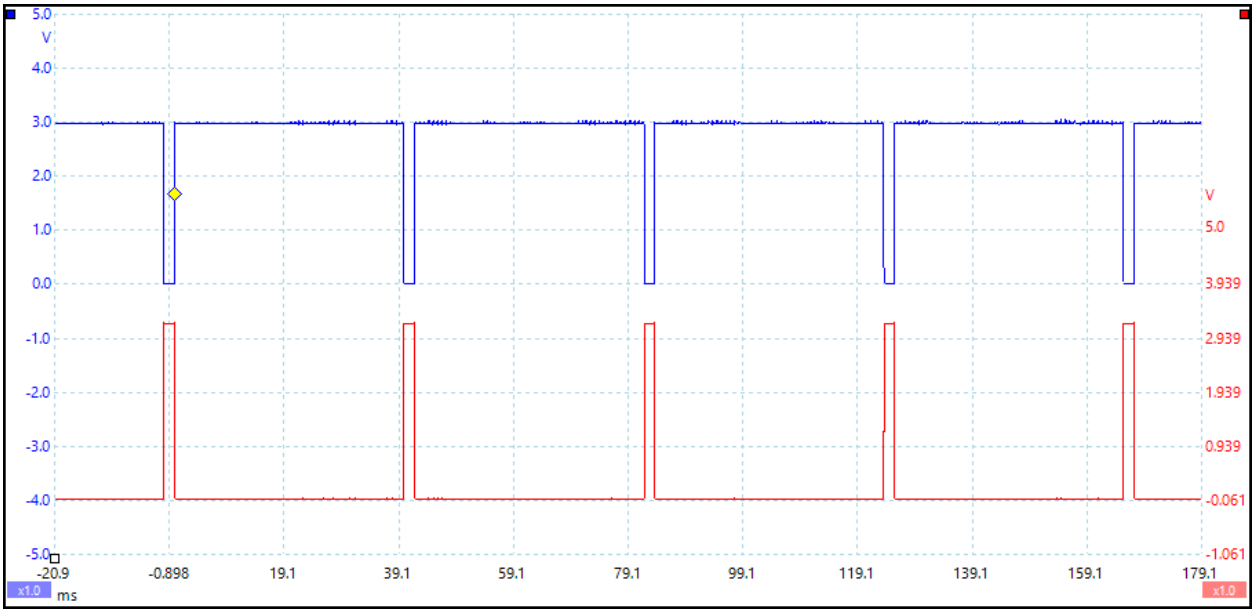


The picture below shows the GPIO port on the Omni 60. The connector on the camera is Molex 0015912035, and it mates with Molex 0050579403. The pin closest to the USB3 connector is camera GND, and the pin on the opposite side of the connector is input. The input is buffered and accepts a wide voltage range of 2 to 24V, however note that none of these pins are opto-isolated in the camera.



The sensor will trigger on the falling edge of the signal fed to the GPIO port. Internally the mt9v022 sensors trigger on the rising edge of the trigger signal, however since the GPIO circuit inverts the signal you must generate an inverted signal. Below, the pulse train fed to the GPIO port is in blue, and the internal inverted signal that is seen by the sensors is shown in red. The sensors will trigger on the rising edge of the red signal, and on the falling edge of the blue signal (the signal you are providing).

The pulses do not have to be evenly spaced, and can be at arbitrary frequency or phase, given that frequency does not exceed 30 FPS. The 30 FPS limit is a limitation of the mt9v022 sensor itself.



Once you are providing the correct trigger pulses on the GPIO input pin, the next step is to put the five sensors into snapshot mode. In snapshot mode, the sensor triggers frames individually based on the trigger input provided from the GPIO port. In master mode, which is the default, the sensor is free-running and triggers its own frames at the maximum rate possible.

Configuring the sensors to use snapshot mode is done by setting bits 3 and 4 on the 0x7 register of the five sensors:

```
occamWriteRegister(device, 0xcc02, 0x1);
occamWriteRegister(device, 0xcc01, 0xb8);
uint32_t value = 0;
occamReadRegister(device, 0x7, &value);
value |= 1<<3;
value |= 1<<4;
occamWriteRegister(device, 0x7, value);
```

Note that the frame rate will drop to zero unless you are providing a trigger input to the camera.

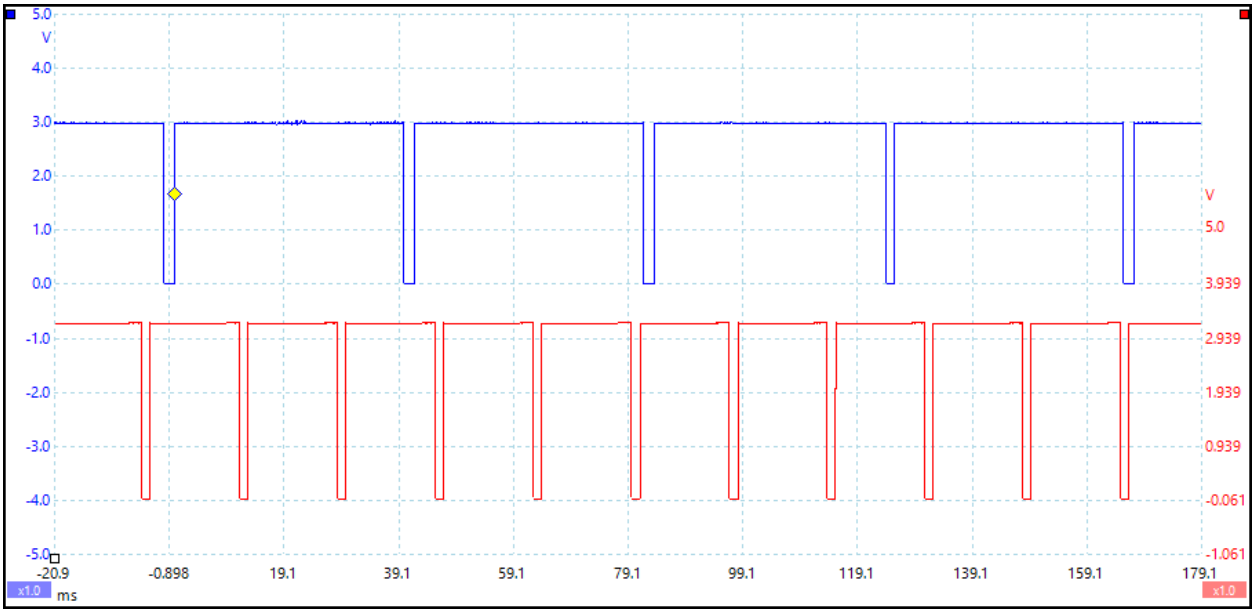
To revert to master mode, you can unset bits 3 and 4 of register 0x7 as follows:

```
occamWriteRegister(device, 0xcc02, 0x1);
occamWriteRegister(device, 0xcc01, 0xb8);
uint32_t value = 0;
occamReadRegister(device, 0x7, &value);
value &= ~(1<<3);
value &= ~(1<<4);
occamWriteRegister(device, 0x7, value);
```

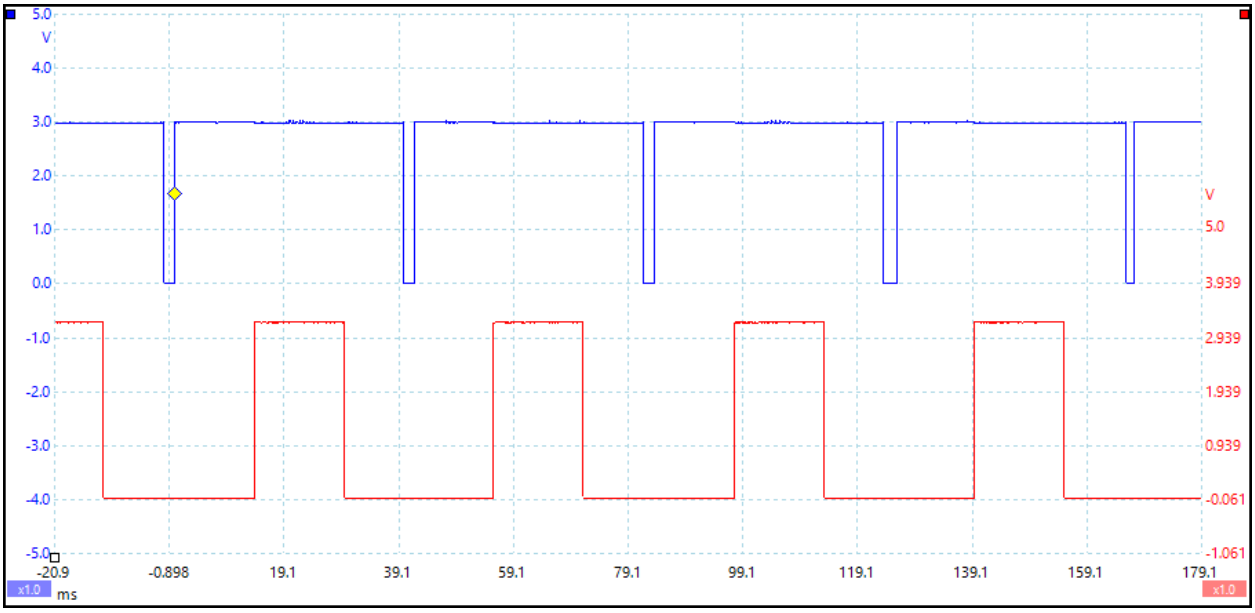
You can also power cycle the camera to revert to master mode, since this is the default.

Below, the blue signal is the trigger pulse from the GPIO port, and the red signal is the frame_valid output signal of the sensor. This signal rises when sensor readout starts and falls when readout ends. Download to the host starts on the falling edge of frame_valid and is buffered in framebuffer memory on the camera during readout.

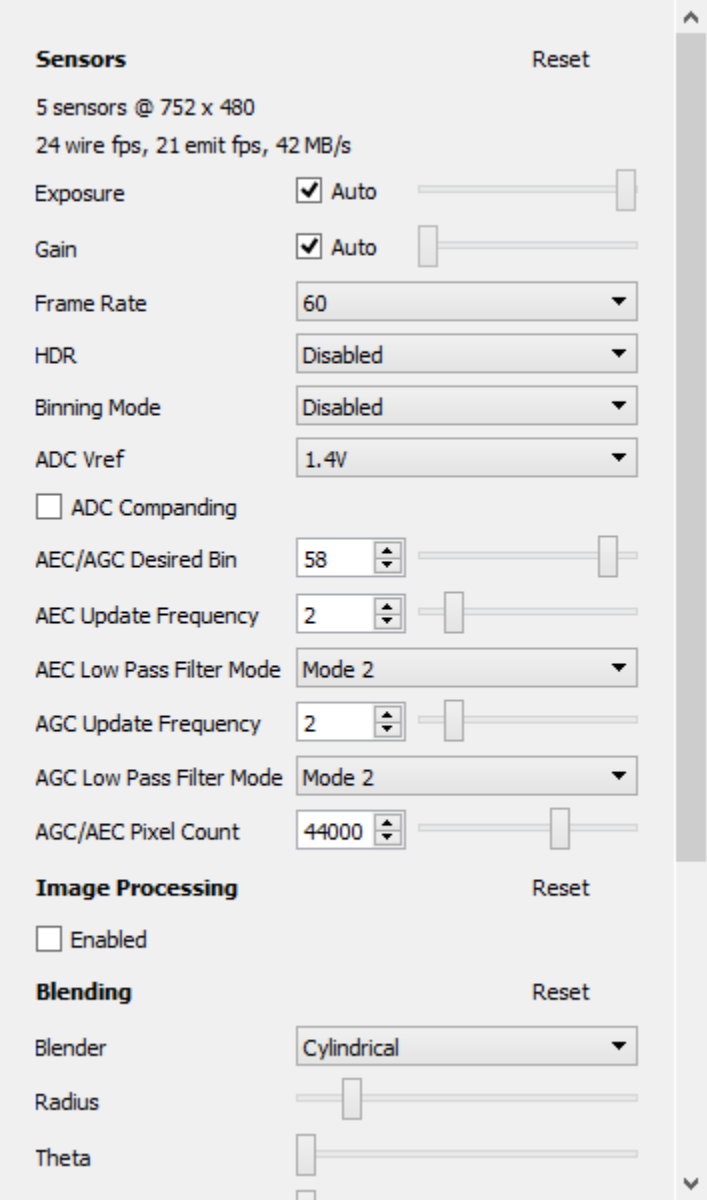
When operating in master mode, the frame_valid signal shown in red is at different frequency (60 FPS below), phase offset, and drifting from the trigger signal.



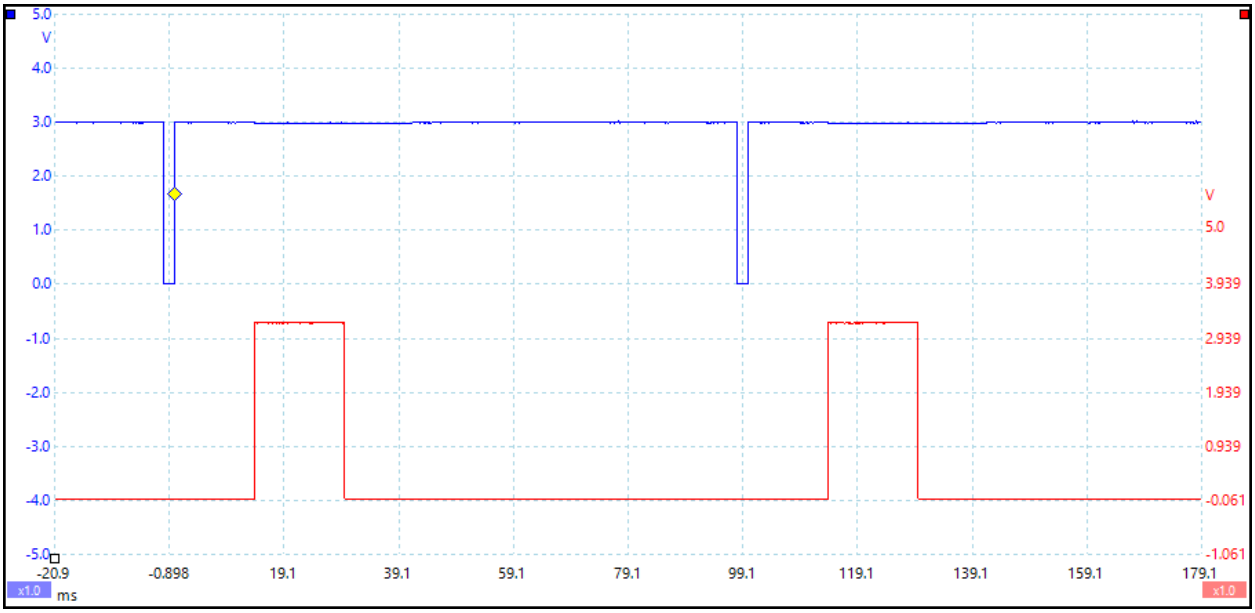
Once the sensors are placed into snapshot mode, the frame_valid readout signal follows the trigger pulse. Here, the trigger is running at 24 Hz:



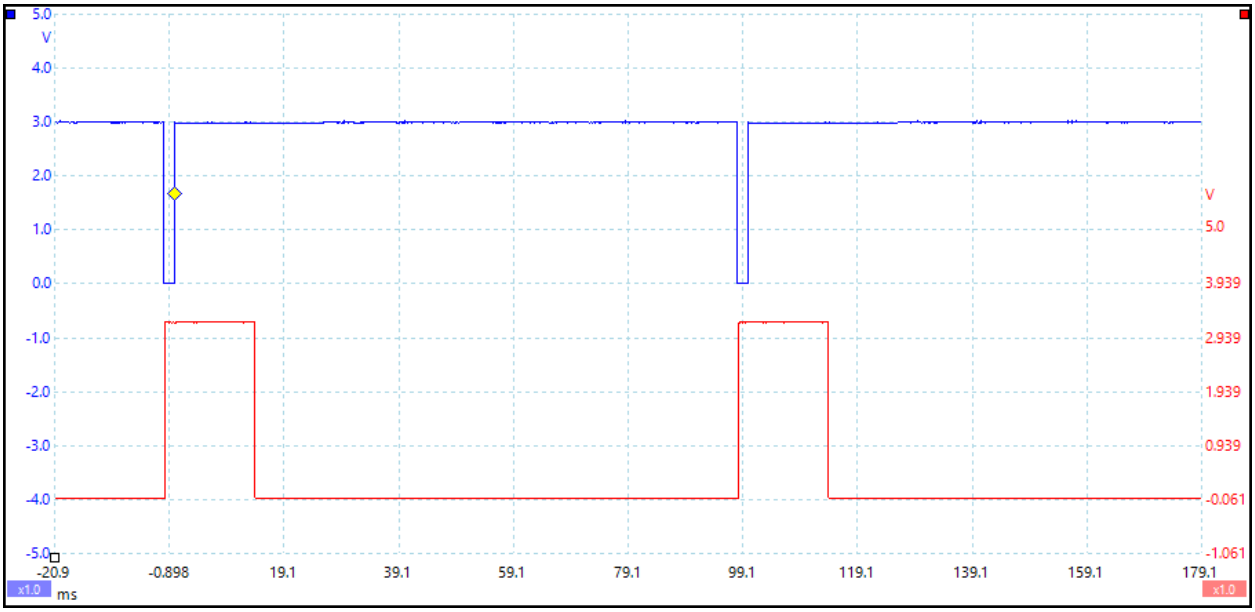
The reported “wire fps” reported in the software should match the trigger pulse frequency, as in the following picture:



Lowering the frequency of the GPIO trigger strobe to to 10Hz, you can see that the sensor readout matches. Note that the red signal in these graphs show sensor readout (readout occurs while red signal is high). Exposure of each frame starts precisely at the GPIO trigger, and after the configured exposure period, the readout begins. The graph below is showing a ~16 ms exposure duration.



This graph is showing the minimum exposure duration:



CHAPTER

TWELVE

MOUNTING

The Omni 60 and Omni Stereo cameras are both mountable via a single 1/4”-20 bolt on the bottom of the device. The bolt is located in the center of the circle touching the outermost corners of the device (i.e., the center of the bottom face of the device).

Please refer to the mechanical drawing of the camera for more detailed information.

CHAPTER

THIRTEEN

TROUBLESHOOTING

13.1 Pictures appear but with lag

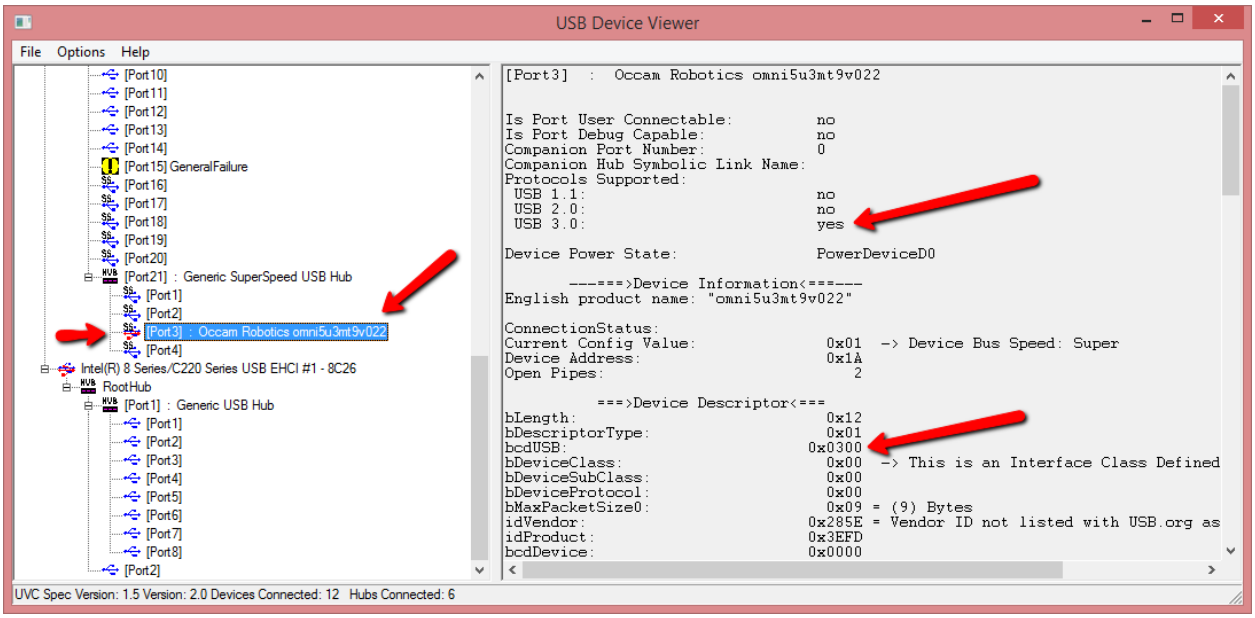
This is a problem that has been reported for users using the Omni 60 camera with a USB2 connection. In firmware versions <= 1.2.6 the device kept 4 frames in circular buffer on the device, and if the host PC is not able to download fast enough this will manifest in a latency between the time images are captured and when they are displayed on the screen.

It is highly recommended that you are using USB3, since the camera requires the higher throughput of USB3 for best performance. USB2 bulk transfer rate is under 60 MB/s, whereas the Omni 60 video stream requires around 104 MB/s.

If for your application you must use USB2, please check for more recent firmware for your camera from the Downloads area. In recent firmwares the number of frame buffers kept on device is only two (to ping pong between capture and download to host), so the latency will be lower. If you are running firmware 1.4.2 or greater, you can also use binning mode to reduce the amount of data transferred to the host by 2x or 4x.

USB3 ports appear with blue connector on the PC side. Here is what a USB3 connector looks like on virtually all computers:

You can also download `usbview.exe` which is a developer program from Microsoft that allows you to see whether your device is enumerated properly as USB3.



13.2 Linux: camera appears in lsusb but SDK demo programs don't detect it

This is caused by the linux/udev policy that unknown USB devices get enumerated and owned by the superuser by default. The Indigo SDK uses the userspace libusb library to access the camera, which requires that the device file under /dev/bus/usb for the camera is read/write-able by the user running the SDK demo programs.

You need to tell udev to allow read/write access to the device for all users. To do this, create a file /etc/udev/rules.d/occam.rules that contains the following line:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="285e", ATTRS{idProduct}=="3efd", MODE="0666"
```

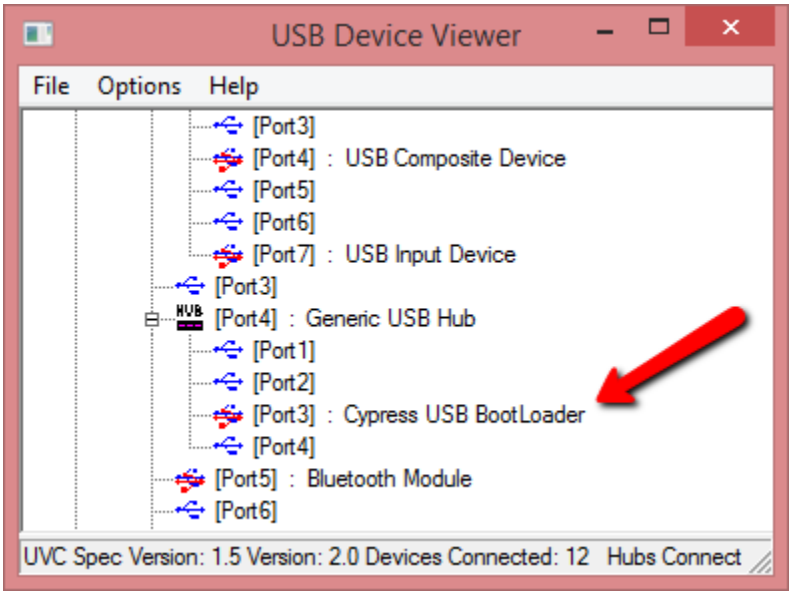
If you don't want to modify udev for whatever reason, you can manually sudo chmod 666 /dev/bus/usb/<bus>/<index> to allow read/write access to all users. <bus>/<index> can be found via lsusb. Also you can run the demo programs under sudo, of course, though that is not recommended.

13.3 Camera is bricked after failed firmware upgrade

In the event that after a firmware upgrade your camera no longer comes up in the SDK or the Indigo Tools, it may be that the image did not properly flash (due to being interrupted or another issue).

If the camera gets into this state, you can send the camera back to us and we'll reflash it with a correct image. This article gives a brief walkthrough of how you can get proper firmware back onto your camera yourself.

The firmware is programmed to non-volatile flash memory on the device. On boot (when you plug in the camera) the default ROM firmware loads the image from the flash memory. If a valid image is not able to be read, the camera will come up with just the ROM image. When the camera is in this state, it will enumerate as "Cypress USB BootLoader" in `usbview.exe`, like this:



- Step 1

The main I/O controller on the device is the Cypress FX3 chip. You need to download the development tools for it from their web site here. In particular you'll need to install their cyusb driver and get the "USB Control Center" application installed.

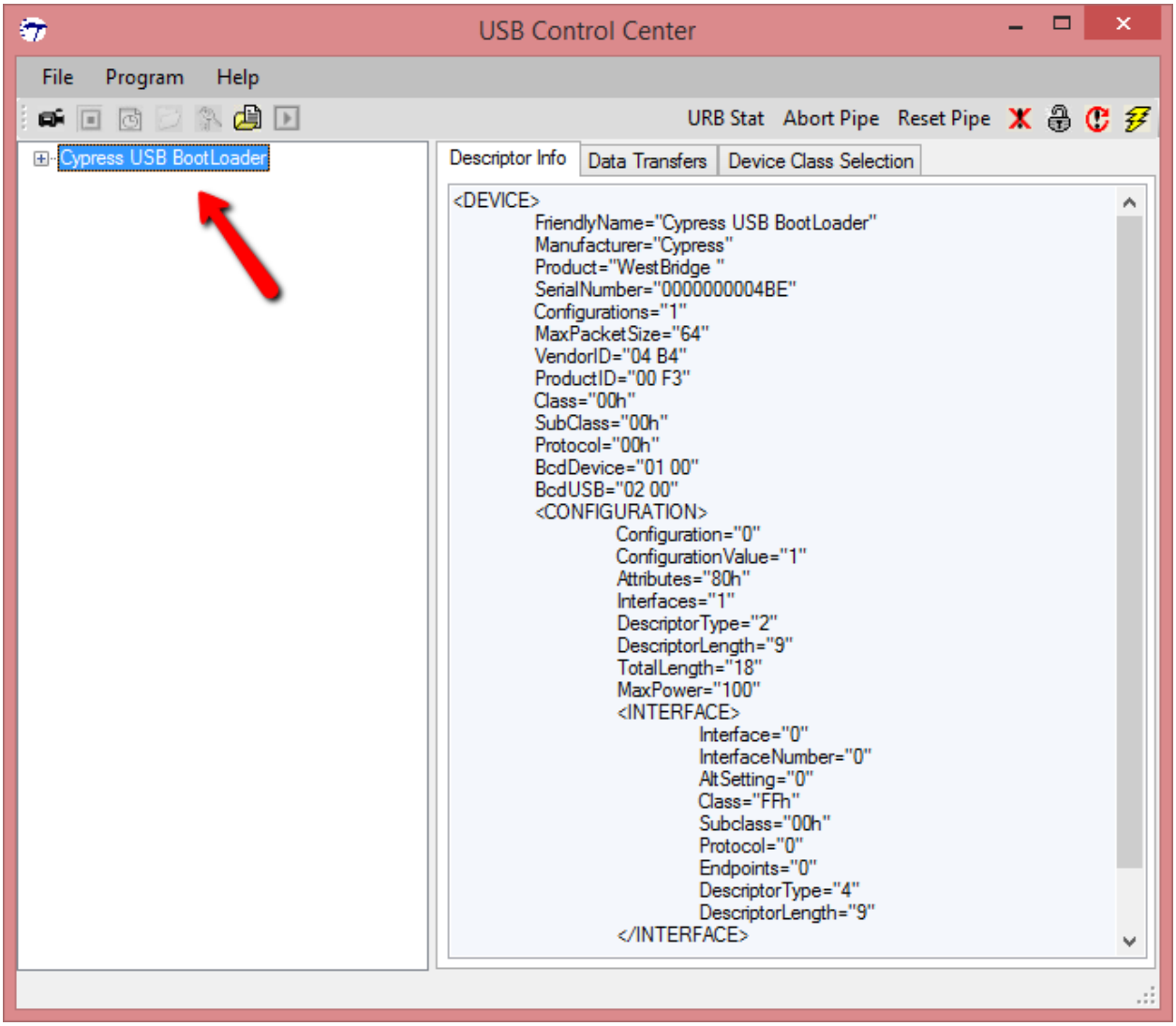
- Step 2:

Download a firmware image from the firmware download area that corresponds to your device. For paired devices such as Omni Stereo, you should download the firmware that corresponds to your base device (for Omni Stereo this is omni5u3mt9v022).

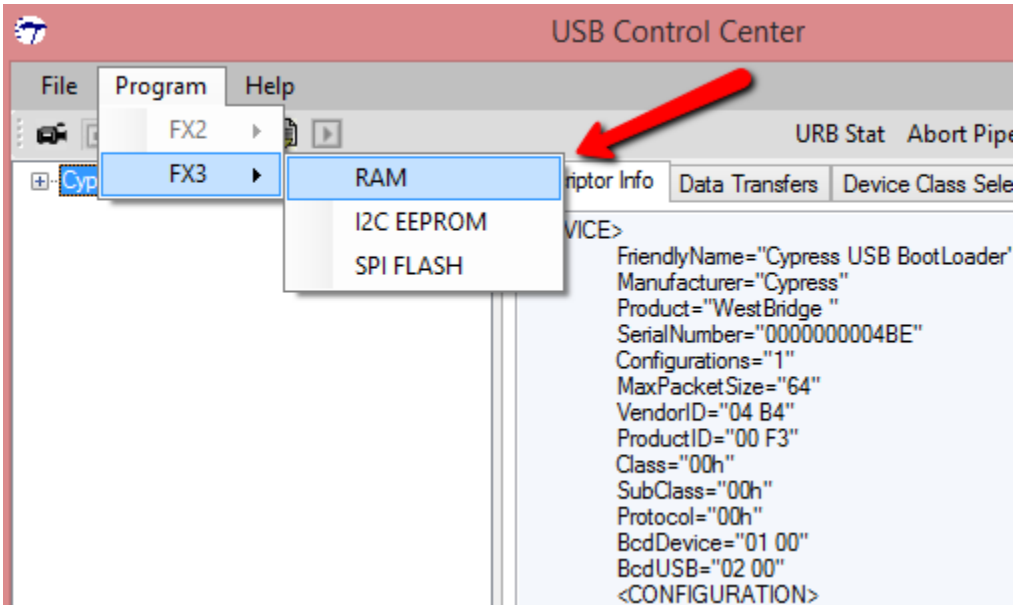
- Step 3:

Open the USB Control Center app that is part of the Cypress FX3 SDK. The device should appear in the enumerated list of devices. If it doesn't, check that it's listed in usbview.exe as shown above. If it's not listed in usbview.exe then it's not plugged in or possibly a hardware issue. If it's listed as "Cypress USB BootLoader" in usbview.exe and doesn't appear in the USB Control Center app, then it's a driver issue. Make sure you have installed the cyusb driver from the Cypress FX3 SDK download page linked above.

Select the device in the list:

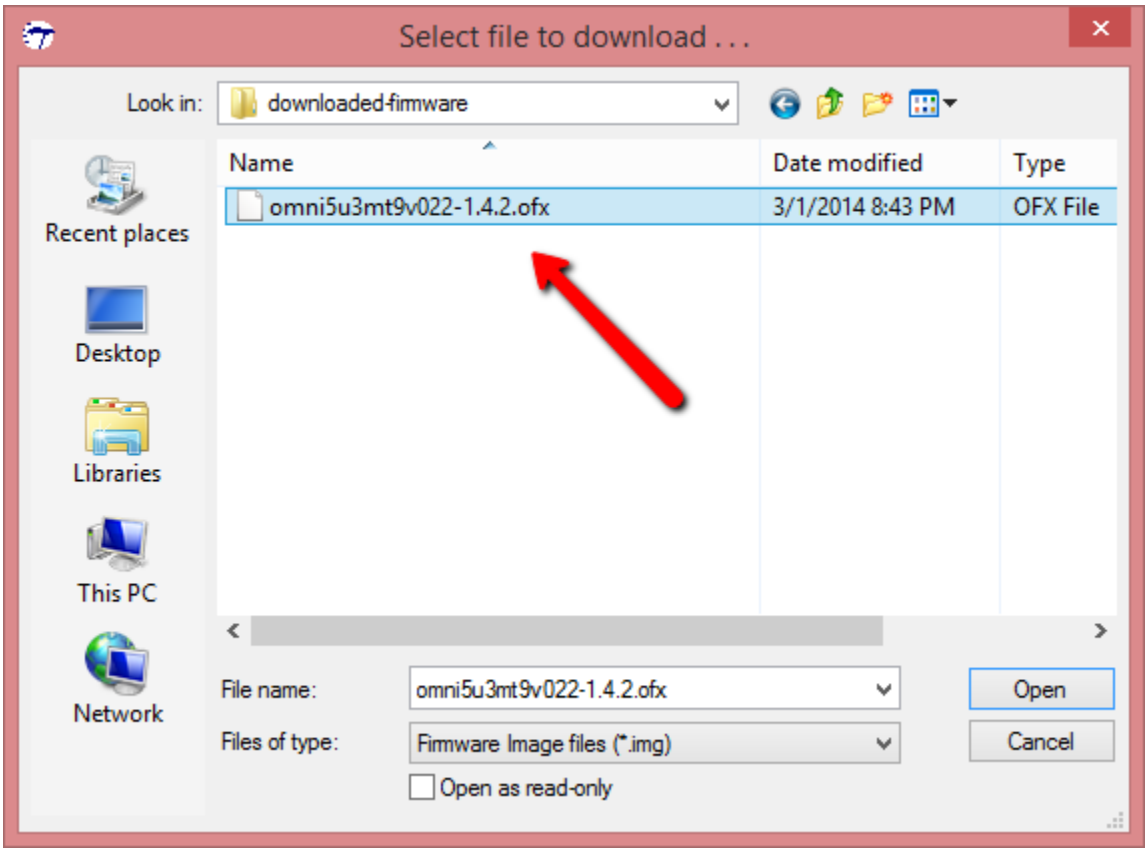


Then select Program | FX3 | RAM from the menu:



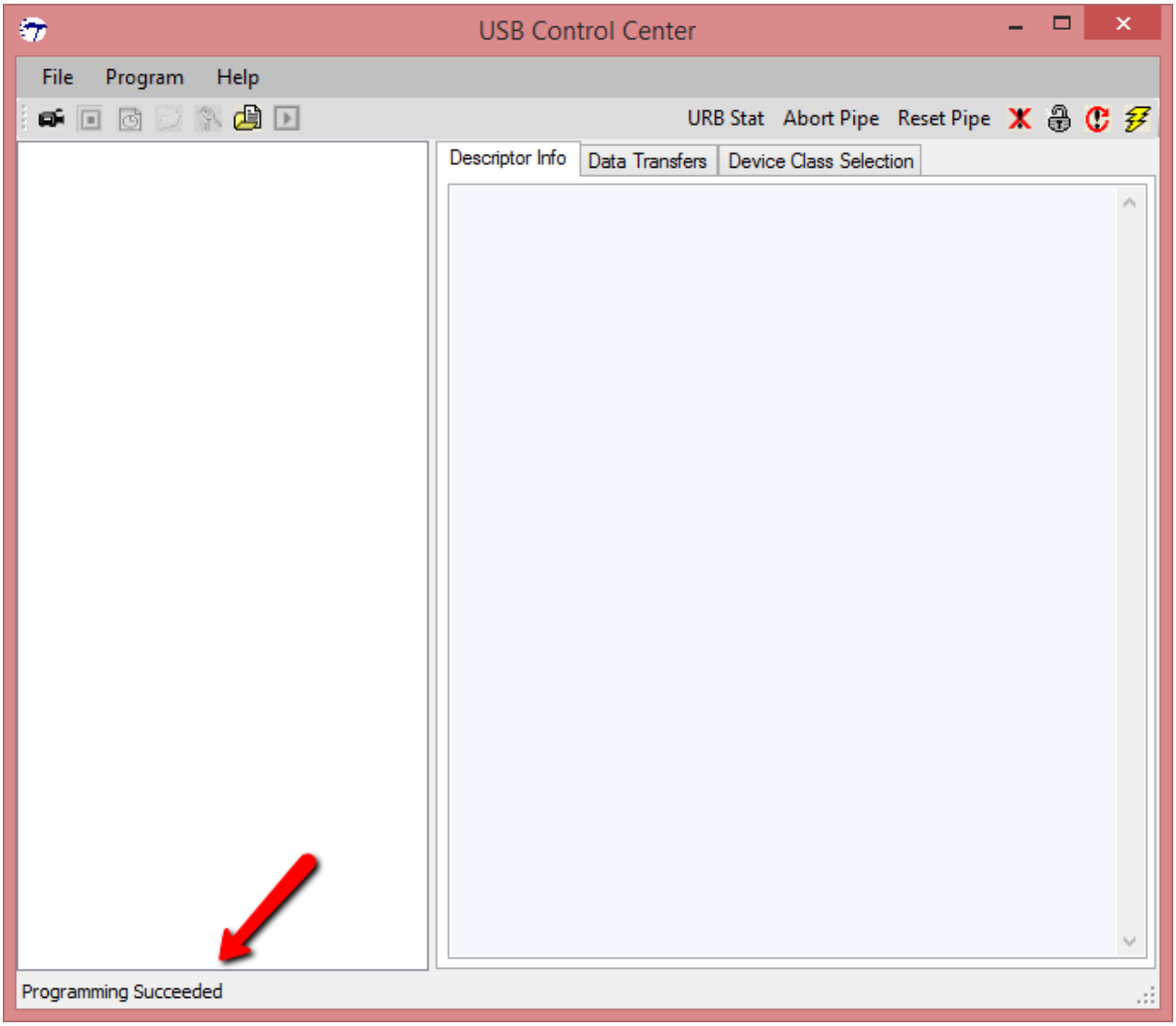
- Step 4:

Select the firmware image you downloaded.



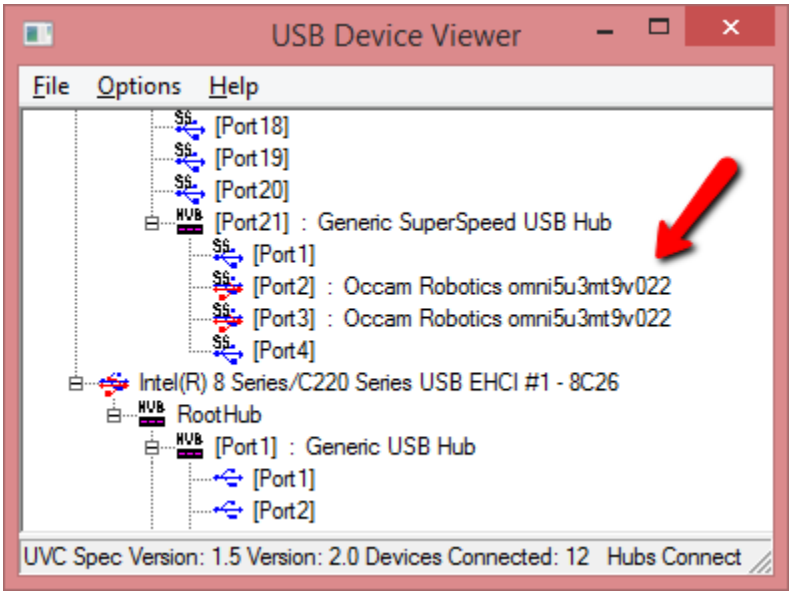
- Step 5:

After you click Open the device's RAM will be programmed with the given image. You should see it say "Programming Succeeded" in the status bar:



- Step 6:

Now the camera should open in Indigo Tools, and from usbview.exe should be listed as an Occam device:



If the device appears properly in usbview.exe but does not appear in Indigo Tools, it's a driver install issue. Please go to the driver download area and make sure the driver is installed properly.

- Step 7

Since we selected to upload the image to RAM, it will lose the image when you power cycle the camera next. You should open Indigo Tools and upgrade the firmware again using the tool in Indigo Tools. Instructions for using the firmware upgade from Indigo Tools can be found here.

You can also use SPI FLASH programming option from the USB Control Center app, but it intermittently fails due to using a higher SPI data rate so it may take a couple of tries.

13.4 Additional Help

If you are having any issues that you can't resolve with this document or others on the site, please email us at support@occamvisiongroup.com for further assistance.

CHAPTER

FOURTEEN

FREQUENTLY ASKED QUESTIONS

14.1 Does the color conversion process happen on the camera or the host computer?

It happens on the host computer, and can be optionally bypassed if recording raw data for offline processing through the SDK pipeline or the users's pipeline is desired.

14.2 In Omni Stereo, does the stereo compute happen on the camera or the host computer?

It happens on the host computer. The various outputs required to perform stereo matching are also exported individually, such as the images themselves, the internal and external parameters of each sensor, transposed rectified images, and disparity images. Internal parameters are radial/tangential model lense parameters, external parameters given sensor pose within the device.

14.3 Can I make my own Omni Stereo with two Omni 60?

You can combine two Omni 60 into an Omni Stereo by using the pairing function in Indigo Tools. This will modify the EEPROM on the device to indicate that each camera becomes half of an Omni Stereo. Then when they are used together, they will be hardware synchronized automatically over USB.

Note that to use the calibration tools and the stereo matching pipeline that work with a normal Omni Stereo, the relative placement of the two Omni 60 should be approximately the same as the normal Omni Stereo. That is, the camera should have the same rotation about the Y axis (i.e., USB connector of top camera should sit above the USB connector of the bottom camera), and the spacing between the cameras should be approximately 12 cm as this is assumed in the calibration tool within Indigo Tools. Please contact us if you would like to use a different baseline.

14.4 Is Mac OS Supported?

No.

14.5 Is a GPU required for either Omni 60 or Omni Stereo?

No, all the work is performed on the CPU.

14.6 Can I retrieve just a subset of the disparity images?

Yes, see the usage of the `occamDeviceReadData` API call. You can request any subset of the device outputs, including disparity images.

14.7 Can I retrieve the raw unstitched sensor output?

Yes, by calling `occamDeviceReadData` with outputs `OCCAM_RAW_IMAGE0` through `OCCAM_RAW_IMAGE4` for Omni 60 and `OCCAM_RAW_IMAGE0` through `OCCAM_RAW_IMAGE9` for Omni Stereo.

There are examples in the SDK `examples` folder showing how to capture raw data and perform the processing pipeline offline using the SDK.

14.8 Is infrared visible in the camera output?

The monochrome version of the camera has good response in the NIR range, specifically about 30% response at 850 nm. See the quantum efficiency graphs associated with the sensor above.

14.9 Can I use an another stereo algorithm?

Yes, by requesting the subset of data you need from the SDK, such as the rectified images, and then running any other stereo algorithm.

14.10 Can I record raw camera data and do the processing offline?

Yes, please see the examples that do this in the SDK `examples` folder.