FLIR ADK

Getting Started

Getting Started guide for the FLIR ADK with USB, GMSL, and Ethernet Connector Options

FLIR Systems OEM & Emerging
6769 Hollister Avenue
Goleta, CA 93117
Phone: +1.805.964.9797
www.flir.com

Document Number: 102-2013-105
Version: 140
Issue Date: June 2020

Table of Contents

1 Introduction

1.1 Revision History

Version	Date	Comments
100	05/02/2019	Initial Release
110	7/9/2019	16-bit Video + cable spec
120	10/14/2019	Ethernet Description
130	11/15/2019	GMSL Description + USB Description
140	6/15/2020	Latency, Intrinsic Calibration, Ethernet API

1.2 Reference Documents

Ref Number	Document number	Comments
1	102-2013-40	Boson datasheet

1.3 SCOPE

This document describes detailed instructions on the conditions and requirements for integrating the FLIR ADK with various communications methods including USB, Ethernet, GMSL accessory board for communication, video streaming, and hardware synchronization.

1.4 BACKGROUND

The FLIR ADK provides a Long Waver Inferred Image (LWIR) that is 640x512 pixels. It senses thermal radiation with wavelengths of 8-14um. The output is either a 8bit or 16bit monochrome image.

The ADK comes in three different interfaces: USB output, GMSL output, and GMSL output with a converter for GMSL to Ethernet. The ADK also has four different lens options.

Horizontal FOV	Vertical FOV
24	19.2
32	25.6
50	40
75	60

2 USB – Getting Started

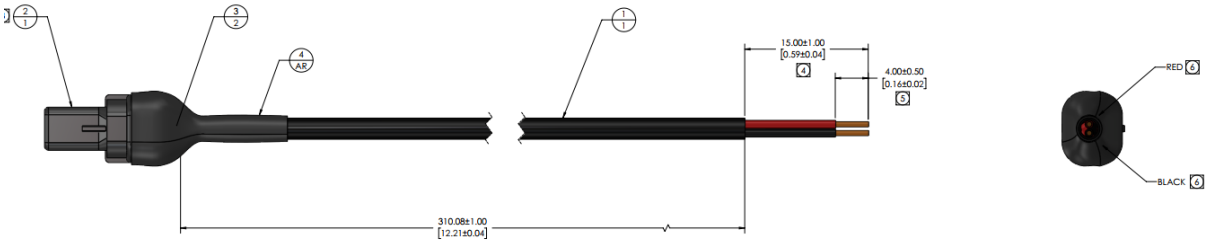
The USB option for the ADK ships with a Boson connected by USB for camera power, video and command and control, a BNC cable for external sync input, and a Mizu-P25 connector to power the window heater.

When connected to your computer, the USB ADK shows up as two different devices. The first is a USB Vision Camera used for streaming video. The second is a com port used for command and control. Any webcam driver should be able to access the USB Vision camera device (Windows10 'Camera', MacOS Photobooth, and Linux Cheese application will all stream the ADK).

2.1 Connect and power the USB ADK

1) Connect the USB cable to the computer.
2) Optional: To power the window heater, apply 12 VDC up to 0.5A to the MIZU-P25 connector (shown below) to power the heater. The heater is protected against reverse polarity so it's technically ok to hook it up in reverse, it will just pull a little more power. The USB ADK ships with a MUZU-P25 pigtail. Connect the green lead of the provided cable to positive and

the black lead to ground.



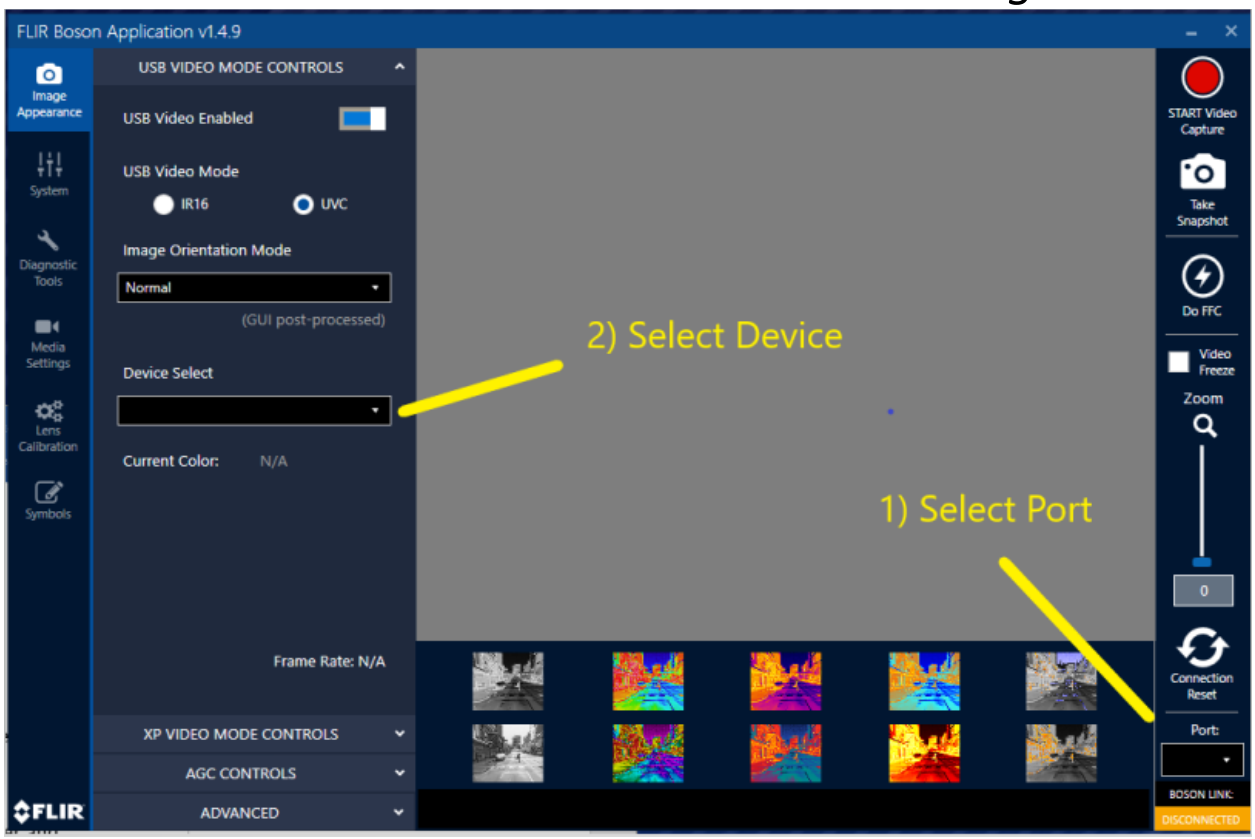
2.2 FLIR Boson GUI

An example GUI for streaming images and controlling the camera can be found here

https://www.flir.com/support-center/oem/camera-controller-gui-for-boson/

After the software is installed find the camera comport by opening Device Manager and listing the available ports. Do this with the camera unplugged and with the camera plugged into the computer. The port that shows up with the camera plugged in is the camera's comport.

With the camera's comport known, open the BosonGUI software (icon is typically installed on your desktop) and select the camera's comport in the Port selection dropdown in the lower right corner of the BosonGUI software. To start video streaming select "FLIR Video" from the "Device Select" drop down menu on the left of the screen.



Once connected, click the "Do FFC" button on the right of the screen. The camera should click and the image should blink. This will confirm you have control of the camera and are streaming video.

There are many nested menus on the left side of your screen. These give you control of the camera. The control options of most interest for automotive customers are:

- "USB Video Mode" under Image Apperance -> USB Video Mode Controls
- "FFC Mode" under System -> FFC Controls. Manual mode is recommended for automotive applications.
- "Table Switch" under System -> Dynamic Range Control. This may be need is the camera sees a significant change in temperature.
- "Save Power On Defaults" and "Restore Factory Defaults" under System -> Configuration control. This lets you boot into a custom camera configuration.
- "TFPA" under Diagnostic Tools -> Status Panel. TFPA stands for Temperature of the Focal Plane Array. This is a measure of temperature inside the camera. On the same menu you can check your camera software version and serial numbers.
- "File Save Path" under Media Settings -> Media Configuration. These show the location of the saved files when using the Video Capture and Take Snapshot buttons in the upper left.

2.3 External Sync

The BNC coaxial connector on the USB ADK cable is for optional external sync. The ADK triggers on the falling edge of a 3-12 volt signal with a minimum pulse width of 90nS. The sync pulse should be 55-60hz even if the camera is in 30hz output mode. The camera will need to be configured for external sync. Failure to provide sync pulses to a camera configured for external sync may crash the camera requiring a reboot.

See the Flir Boson External Sync application note (102-2013-100-04 Rev 150) for more information.

3 Ethernet – Getting Started

The Ethernet option of the FLIR ADK ships with a GMSL connector on the camera and a GMSL to Ethernet Bridge that allows users to interact with the FLIR ADK using the GeniCam protocol.

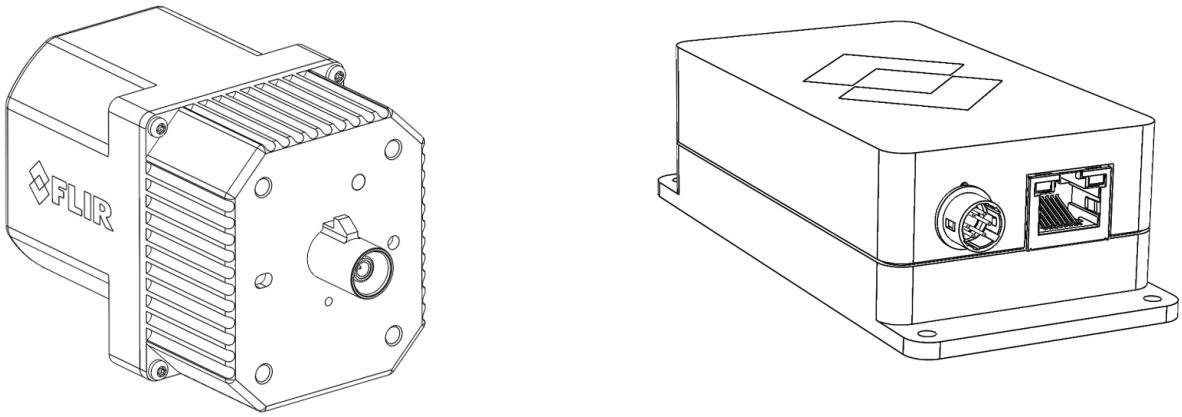


Figure 1: The FLIR ADK GMSL option is shown on the left with the GMSL 1 to Ethernet bridge shown on the right.

The Ethernet Bridge (also known and the Break Out Box or BoB) is a GeniCam GigE Vision device. Any GeniCam driver should work. We offer the Spinnaker ADK as FLIR"s supported GeniCam Client. While not supported we have heard that the open source Aravis driver works as well.

Spinnaker can be downloaded from: https://www.flir.com/products/spinnaker-sdk/

3.1 Connecting the Boson ethernet ADK to the computer

1. Connect the FAKRA cable from the FLIR ADK to the GMSL – Ethernet bridge.
2. Connect the ethernet cable to a network interface card on the host computer.

3. Connect the 6 pin GPIO cable to the GMSL Ethernet bridge.
4. Apply power to the 6 pin GPIO cable. Apply 8-16 VDC to the Green wire and ground on the Black wire. The power supply should be rated for a 2 A peak current. The system is designed to run off the 12V supply from a car battery.
    a. An optional AC power supply that will run the camera and shutter but not the heater is available at: https://www.flir.com/products/12v-power-supply-with-6-pin-hr10-connector/

Table 1: Pin out of the GPIO pins on the Ethernet breakout box. The Colors correspond to the GPIO cable that came with the ADK.

Color	Pin	Function	Description
Green	1	VExt	Camera Input Power
Black	2	Not Connected	-
Red	3	Not Connected	-
White	4	Not Connected	-
Blue	5	Not Connected	-
Brown	6	GND	Camera Power Ground

3.2 Ubuntu Installation

Spinnaker is available for Ubuntu 16.04 and 18.04.

1. Download the appropriate version of Spinnaker SDK from

   https://www.flir.com/products/spinnaker-sdk/

2. Unpack the tar.gz file for the appropriate system architecture.
3. Follow the README directions to install **necessary dependencies** and then run the **installer script**.
4. It's necessary to configure the ethernet card that the Boson is plugged into. It might be necessary to find which ethernet card the Boson is plugged into. To do this run 'ifconfig' with the ethernet cable unplugged then reconnect the ethernet cable and run 'ifconfig' again. You should see a change in one of the ethernet cards.
5. Now configure that ethernet card by opening the network interfaces file with your favorite text editor.

```
sudo gedit /etc/network/interfaces
```

Add the following lines (make sure to change enp15s0 to the appropriate card name found by running ifconfig)

```
iface enp15s0 inet static
address 169.254.0.64
netmask 255.255.0.0
mtu 9000
auto enp15s0
```

Lastly restart the networking service to apply settings

```
sudo /etc/init.d/networking restart
```

6. The installer installs a program called SpinView. This allows you to see live feed video from the camera. To start SpinView enter 'spinview' into the terminal.
7. The Linux installer also provides example programs in /usr/src/spin naker/src and full documentation in /usr/share/doc/spinnaker-doc.
8. When SpinView is open the Boson device should be visible under the devices tab (as shown below). Click on the Boson device to select the camera and the click the play button to display a live feed from the camera.
9. Similar to the USB GUI, spinview provides access to the Boson features in the Features tab. You may need to stop streaming video to change some settings.

If you notice that some images are getting dropped or that the video feed is choppy it may help to increase the Rx and Tx buffer size. Below is a link describing how to do that.

https://www.itechlounge.net/2015/05/linux-how-to-tune-up-receive-tx-and-transmit-rx-buffers-on-network-interface/

3.2.1 Robot Operating System

We have an example project that uses the Ethernet ADK in the ROS framework present here.
https://github.com/flir/flir_adk_ethernet

Once the Boson is setup to run with Linux then follow the instructions in the Readme.md to get the Boson to work with ROS.

After setup you should be able to run which will stream video on the host computer.

```
roslaunch flir_adk_ethernet boson.launch
```

3.3 Windows Installation

Download and install the Windows version of the Spinnaker SDK here.

https://www.flir.com/products/spinnaker-sdk/

The Spinnaker SDK also comes with a software application called SpinView which will display real time images from the Boson Camera.

3.4 Working with the Spinnaker API

When using the Ethernet BoB with the Boson camera, there is no direct access to the camera. All communication with the camera is implemented with the Spinnaker API. When connecting to the camera, the camera provides a list of features. The API provides a way to access those features.

You are strongly encouraged to review, run and modify the example programs. On Linux they are installed in /usr/src/spinnaker/src

3.4.1 Accessing Boson Camera Feature Example

This example show how to execute a manual FFC (also known as shutter) with the Spinnaker API. This code was placed just before the 'Begin acquiring images' comment in the AcquireImages function of the Acquisition.cpp Spinnaker example.

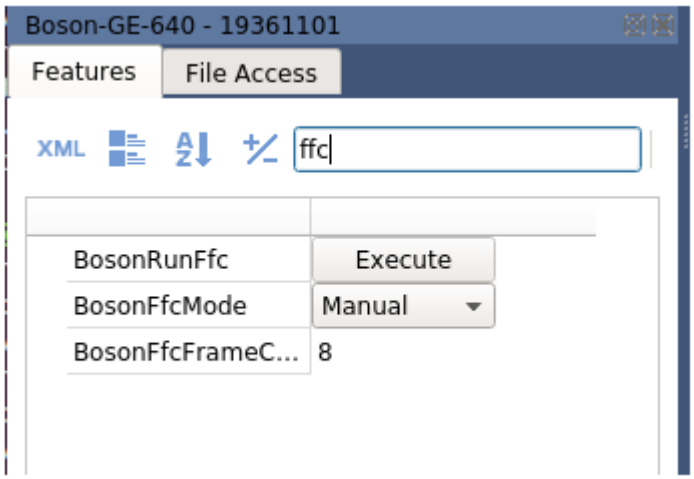
```
CCommandPtr ptrRunFFC = nodeMap.GetNode("BosonRunFfc");
if(IsAvailable(ptrRunFFC) && IsWritable(ptrRunFFC)){
    cout << "Running FFC" << endl;
    putRunFFC->Execute();
} else {
    Cout << "Unable to access FFC" << endl;
}
```

In this example, we ask for the 'BosonRunFFC' node from the node map. Make sure that node is available and that we can write to it. Then we execute the nodes function. This node represents a command. Other node may represent strings, integers, floats or enumerations.

3.4.2 Finding Node Names, Types and Enumeration Values

The above example is good, but how do you know there is a node named "BosonRunFfc". There are two ways to look these up.

The first is using the search features text box in spinview.



This shows a search for "ffc" which stand for Flat Field Correction which is our shuttering process. There are three nodes that contain 'ffc'. Note the displayed name may differ slightly, typically in capitalization or spacing, from the name used in your code.

The formal description of the node is provide with an XML file. When you first access a camera with spinview in Linux, it saves some files to your root directory. The file Boson-GE-640_<serial#>_GenICam.zip describes the features of your camera. Unzipping the file creates the file public_camxml.xml, an unformatted xml file. Use can reformat this file using:

```
xmllint –format public_cmaxml.xml > Boson.xml
```

The resulting Boson.xml file looks like this

```
<Integer Name="BosonSensorSerialNumber" NameSpace="Custom">
  <ToolTip>The serial number of the boson sensor.</ToolTip>
  <Description>The serial number of the boson sensor.</Description>
  <DisplayName>Boson Sensor Serial Number</DisplayName>
  <Visibility>Expert</Visibility>
  <ImposedAccessMode>RO</ImposedAccessMode>
  <pValue>BosonSensorSerialNumber_Val</pValue>
  <Representation>PureNumber</Representation>
</Integer>
```

This snip from the file shows the BosonSensorSerialNumber node. The first line defines it as an Integer node. You would get the node with

```
CIntergerPtr bosonSerialNumberPtr =
nodeMap.GetNode("BosonSensorSerialNumber");
```

The snip also shows the BosonSensorSerialNumber node might be displayed as “Boson Sensor Serial Number” with spaces.

4 GMSL – Getting Started

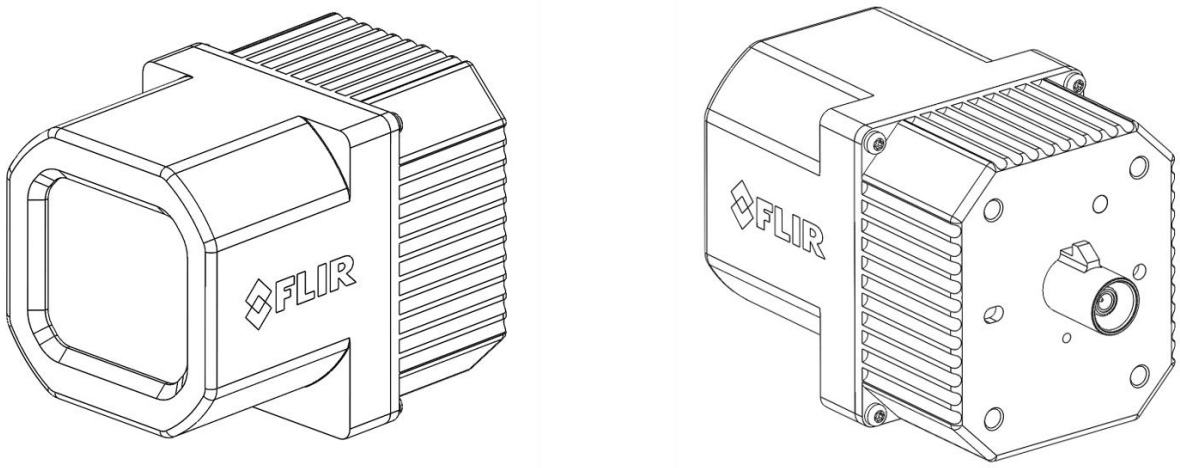


Figure 2: The FLIR ADK with the GMSL configuration.

The FLIR ADK with the GMSL connector allows for the longer cable lengths that are necessary in the automotive environment. The Serializer in the ADK can be configured for GMSL 1 (trailing part number AA1) and for GMSL 2 (trailing part number AA2). Note that Serializers configured for GMSL 1 can be field upgraded to GMSL 2.

In this document we provide examples of how to collect data from the GMSL ADK as well as read and write I2C commands to the ADK. We have a driver for the Nvidia Drive system and we show an example of using the Maxim MAX9296A_CSI_EVKIT to control the ADK. Note that at the time of writing this GMSL 2 is still Maxim proprietary information, thus it is necessary to contact Maxim directly to get documentation about the SerDes chips.

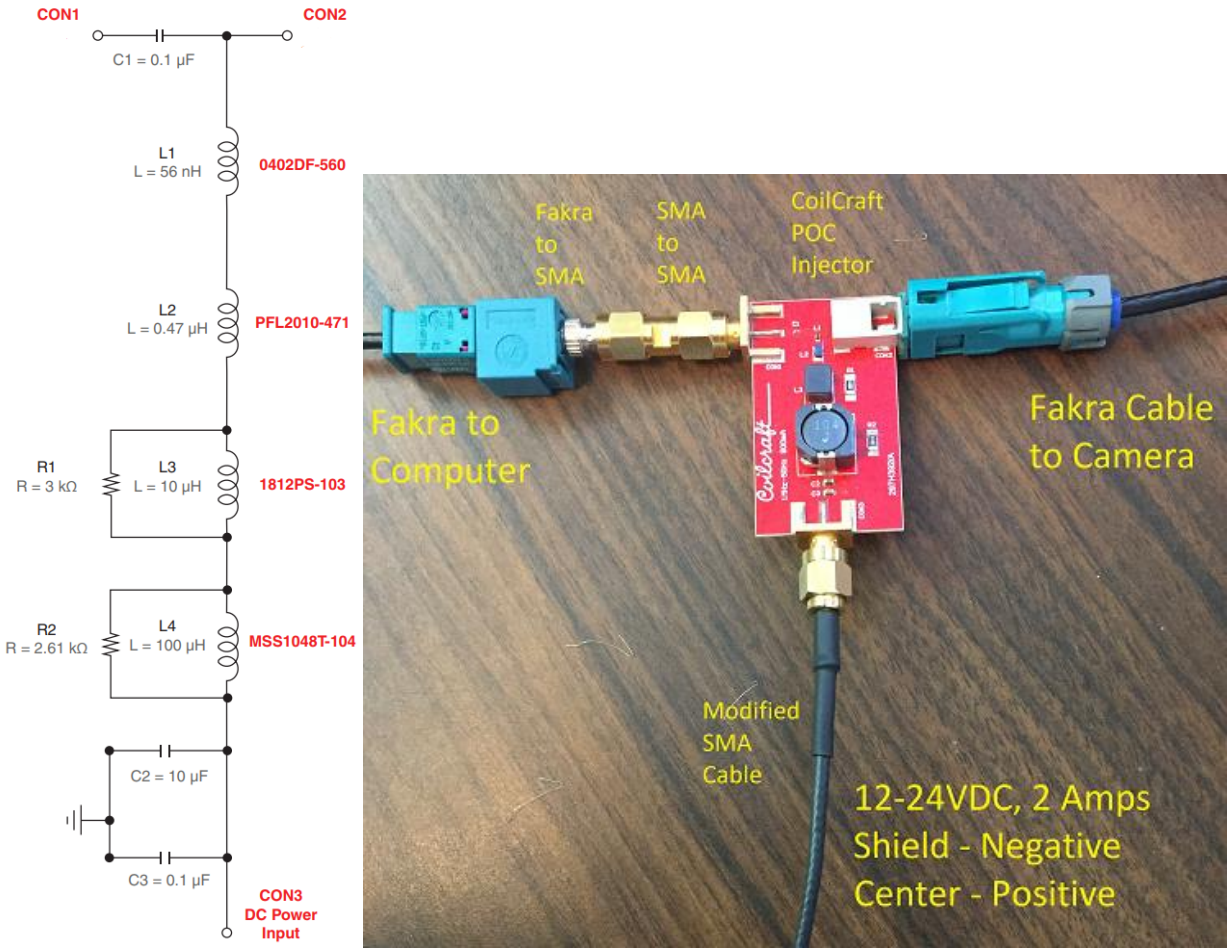
4.1 Power Over Coax

The FLIR ADK features a heater and a shutter that are necessary for getting the most out of the thermal core. The heater and shutter require more power than is typically provided by the GMSL connection on most platforms (including NVIDIA Drive and OnSemi). Many of these systems have a way to change the power injection circuit on the compute platform or

add an external power supply. Please contact your platform manufacturer for those instructions.

FLIR has a recommended power injection circuit you can use to power the ADK without modifying your compute platform using the following components from CoilCraft.

Inductors	DCR max (Ohms)	Notes
0402DF-560 (56nH)	0.061	
PFL2010-471 (0.27 uH)	0.069	
1812PS-103 (10 uH)	0.170	3.0 kOhm resistor in parallel
MSS1048T-104 (100uH)	0.224	2.6 kOhm resistor in parallel



Firgure 3. Power Injector Schematic and Assembled Circuit

For further information and support on PoC including demonstration hardware, please contact Coilcraft at poc@coilcraft.com.

FLIR add the following adapters for the CoilCraft demonstration hardware.

Manufacturer	Part Number	DigiKey P/N
Amphenol RF	APH-FKJ-SMAJ (DigiKey ARF1012-ND)	Adapter Fakra – SMA
Amphenol RF	132168 (DigiKeu ACX1240-ND)	Adapter SMA Plug-Plug
Amphenol RF	135101-02-12.00 (DigiKey ACX1572-ND)	Cable, SMA Plug, R174, 12”

4.2 NVIDIA Drive Systems

FLIR provides a driver for the NVIDIA drive systems that enables the use of the FLIR ADK with the NVIDIA stack. The FLIR driver can be downloaded here.

System	Driver	Notes
Drive PX2	https://flir.box.com/s/kls1q0ll1os462wx6b977f3xt2jd1rac	Not Supported, requites power injector
Pegasus	https://github.com/FLIR/ADK_GMSL_DRIVER Sample Program: https://github.com/FLIR/NVIDIA_IMG_CC_FLIR	Requires Power Injector or HW Modification
Xavier w/ D3 Engineering GMSL SERDES card	https://flir.box.com/s/cqbe4uqa519enox2hmoom3shamlgo5et Sample Program: https://github.com/FLIR/Xavier_OpenCV_Sample	Can’t shutter while heater is running

Follow the directions in the README.md to install the driver and get started with video streaming.

If the NVIDIA system is setup for GMSL 2 the ADK will have to be field upgraded to GMSL 2 for our driver to work. To field upgrade the ADK to GMSL 2 see section 4.3. If you know before hand that your system will require GMSL 2 cameras it is possible to order the ADK configured for GMSL 2.

4.3 Upgrading the GMSL Serializer to GMSL 2

The FLIR ADKs are by shipped in either GMSL1 or GMSL 2 mode. The GMSL 1 cameras have the trailing part number AA1 or AAX and GMSL 2 cameras have the trailing part number AA2. If the camera is in GMSL 1 mode and to be integrated on a GMSL 2 system it will be necessary to configure the Serializer on the ADK to use GMSL 2. This section describes how to configure the serializer on the ADK for GMSL 2.

ADK Serializer Address: 0x84 (8-bit addressing)
Deserializer address is typically 0x90 (Depends on system / deserializer configuration)

1) Configure deserializer for GMSL1 mode (note: in step 4 if you're not able to read back the register from the serializer this is because the deserializer didn't get configured for GMSL 1, a work around to configuring the deserializer to GMSL 1 is to set the CFG pin level of CFG1 to 1. This configures the deserializer to come up in GMSL1 with HIM enabled, this matches the Serializer default configuration.)
   des: write val 0x1F to reg 0x06
2) Turn on local I2C ACK on deserializer (Faked ACK response from the serializer)
   des: write val 0x80 to reg 0xB0D
3) Turn on high immunity mode (HIM), configure HV_SRC on deserializer:
   des: write val 0xE8 to reg 0xB06
4) Turn on CLINK on serializer side (forward channel enabled)
   ser: write val 0x43 to reg 0x404
5) Turn off deserializer local I2C ACK (because we don't need it anymore because forward channel from serializer is now working)
   des: write val 0x0 to reg 0xB0D
6) Turn on Parallel video mode on serializer (may be different for MIPI) (configures forward channel to be run from Boson's PCLK)
   ser: write val 0xF7 to reg 0x07
7) Switch from CLINK mode to SEREN mode on serializer (Now running off Boson's PCLK)
   ser: write val 0x83 to reg 0x404
8) Change from GMSL1 to GMSL2 mode on serializer (we lose communication to the serializer after doing this until deserializer is converted to GMSL2 as well) (upgrade remote side first)
   ser: write val 0x91 to reg 0x06
9) Change from GMSL1 to GMSL2 mode on deserializer (LOCK will be indicated after the link comes back up)
   des: write val 0xDF to reg 0x06

Note: The GMSL 2 configuration on serializer will be lost on power cycle, if running a GMSL 1 camera in a GMSL 2 configuration you need to implement this process each time.

4.4 GMSL SerDes Data Transmission

The Boson core of the ADK is wired to the Maxim Serializer as shown in Table 2.

Table 2: The Boson and Maxim 9295A pin correlation.

Boson	MAX9295A
Data bit 0	D2P, Pin 19
Data bit 1	D2N, Pin 20
Data bit 2	MFP5, Pin 21
Data bit 3	MFP6, Pin 22
Data bit 4	D3P, Pin 23
Data bit 5	D3N, Pin 24
Data bit 6	D0P, Pin 25
Data bit 7	D0N, Pin 26
Vsync	MFP3, Pin 17
Hsync	MFP4, Pin 18
PCLK	MFP0, Pin 2
External Sync	MFP7, Pin 31

4.5 Sending Commands to the ADK over i2C

Currently the Boson camera core of the ADK will only communicate via UART. The GMSL adapter board in the ADK has an i2C to UART FIFO transceiver that will allow you to send byte array commands to the Boson. In practice, the I2C to UART buffer limits the amount of data you can send to the Boson at any one time to 128 bytes. The I2C to UART transceiver has the 8-bit address 0xD8.

The standard cycle for sending I2C commands to the boson is:

1) Turn off the transmitter on the I2C to UART transceiver.
    a. Write value 0x02 to register 0x09 to device 0xD8.
2) Send a series of I2C commands to the FIFO buffer.
    a. For each *value* in command byte array. Write *value* to register 0x00 to device 0xD8
3) Turn on the transmitter on the I2C to UART transceiver – this flushes the FIFO buffer to the Boson.
    a. Write value 0x02 to register 0x09 to device 0xD8.

Note: there are example i2cWrite, i2cRead, and sendCommand functions for a teensy board connected to the Maxim deserializer in Sample I2C Functions For Teensy SOC.

To generate the command byte arrays to send to the Boson, customers can use the rawBoson tool provided through FLIR on GitHub at https://github.com/FLIR/rawBoson combined with the Boson SDK documentation/Software IDD. The Software IDD describe the command

the Boson will accept, and the rawBoson tool can packetize the message properly with a CRC code and start and stop flag. Documentation on how the CRC code and complete bytes need to be created and formatted will be provided at request. We recommend using this tool to determine the corresponding byte array to send for a desired command.

UART TX and RX FIFO are both 128 bytes, so large packets (like chunks of a camera firmware update) can overflow the FIFO if chunk size is not limited to ~64 bytes (in case the FIFO is not cleared between two successive large messages). Most if not all SDK UART commands do not exceed 64 bytes so this isn't a problem in practice except for file transfer. Note it's possible to update the camera firmware over GMSL, it will just take a while to send all the data over to the Boson.

Below is an example of turning the window heater on and off. Note that since the transmitter on the I2C to UART transceiver is not turned off these commands are written directly to the boson.

Turn Heater OFF – the heater is off by default

*Set GPIO state register on ADK side to configure heater **off***
*write val 0x0**1** to reg 0x19 to device reg 0xD8*
Configure heater GPIO as an OUTPUT (step can be skipped if already done this power cycle)
write val 0x0F to reg 0x18 to device reg 0xD8

Turn Heater ON

*Set GPIO state register on ADK side to configure heater **on***
*write val 0x0**9** to reg 0x19 of device reg 0xD8*
Configure heater GPIO as an OUTPUT
write val 0x0F to reg 0x18 of device reg 0xD8

4.6 Sample I2C Functions for Teensy SOC

Below are sample functions for i2cwrite and i2cread as well as a function to send an array of bytes.

I2C Read and Write:

```
void i2cwrite(byte addy, byte reg, byte val)
{
  byte err;

  Wire.beginTransmission(addy);
  Wire.write(reg);
  Wire.write(val);
  err = Wire.endTransmission();
  if (err != 0) {
    I2Cerror = 1;
  }
```

```
}

byte i2cread(byte addy, byte reg)
{
  byte err;
  byte res = 0;

  Wire.beginTransmission(addy);
  Wire.write(reg);
  err = Wire.endTransmission();
  if (err != 0) {
    I2Cerror = 1;
  }

  Wire.requestFrom((int) addy, (int) 1, false);

  while (Wire.available()) { // slave may send less than requested
    res = Wire.read(); // receive a byte
  }
  return res;          // send the byte
}
```

```
byte checkRXbuffer()
{
  //  byte addy = 0x6C;
  byte reg = 0x12;

  return i2cread(transAdd, reg);
  delay(50);
}

byte checkTXbuffer()
{
  //  byte addy = 0x6C;
  byte reg = 0x11;

  return i2cread(transAdd, reg);
}
```

Send I2C Command Array:

```
byte transAdd = 0x6C; // 0x6C - 7bit address. 0xD8 - I2C to UART Chip 8-bit
address


void sendI2CCommandArray(byte commandBytes[], int len)
{
  i2cwrite(transAdd, 0x09, 0x02); // disable the transmitter on transceiver

  for (int i = 0; i < len; i++){
```

```
    i2cwrite(transAdd, 0x00, commandBytes[i]); // write byte to fifo buffer
register on transceiver
  }

  if (debug_messages) {
    DEBUG.print("Message queued: 0x");
    DEBUG.print(checkTXbuffer(), HEX);
    DEBUG.print("\n");
    DEBUG.print("\nCommand Sent\n\n");
  }

  i2cwrite(transAdd, 0x09, 0x00); // enable the transmitter on transceiver.
This flushed the fifo buffer to the boson.
}
```

It's also necessary to write I2C commands to the Serializer and Deserializer that are 16-bit addressed. These are also shown here.

```
void i2cwrite16(byte addy, byte regH, byte regL, byte val)
{
  byte err;

  Wire.beginTransmission(addy);
  Wire.write(regH);
  Wire.write(regL);
  Wire.write(val);
  err = Wire.endTransmission();
  if (err != 0) {
    I2Cerror = 1;
  }
}

byte i2cread16(byte addy, byte regH, byte regL)
{
  byte err;
  byte res = 0;

  Wire.beginTransmission(addy);
  Wire.write(regH);
  Wire.write(regL);
  err = Wire.endTransmission();
  if (err != 0) {
    I2Cerror = 1;
  }

  Wire.requestFrom((int) addy, (int) 1, false);

  while (Wire.available()) { // slave may send less than requested
    res = Wire.read(); // receive a byte
  }
  return res;          // send the byte
```

```
}
```

4.7 Data Format and Boson Settings for GMSL 1

The GMSL option for Boson currently supports both 8-bit and 16-bit data. Of the 24-bit CMOS output from the Boson only the lower 8-bits are physically connected to the Serializer. 8-bit video over GMSL is available using CMOS configurable options that utilize the lower 8 bits (0-7) of the CMOS output. This is the standard 8-bit AGC video data output from the Boson.

For 16-bit video it is necessary to configure the CMOS output to multiplex the lower 8-bits of the CMOS output. To do this the CMOS channel must be put in IR16 video mode and then it must be set to Multiplex or "Double wide" mode. Double wide mode is currently only available on Boson software version 2.0.18283. GMSL ADKs will have this version already installed. If you are buying Boson core, you may need to upgrade your camera.

4.8 Interpreting the Multiplexed/ Double wide IR16 video

The data comes across the GMSL link as a 1280 x 513 frame, where Column 1 contains the top 8 bits of Column 1 of the VGA sensor's first column, and Column 2 contains the bottom 8 bits of the VGA sensor's first column.

Stitching the top and bottom bits will yield the correct values for the IR16 image. The resultant frame should be the resolution of the sensor, 640x512, plus telemetry.

P1[15:8]	P1[7:0]	P2[15:8]	P2[7:0]	P3[15:8]	P3[7:0]	P4[15:8]	P4[7:0]

4.9 Syncing the Boson using GPIO pins of the SerDes

The Boson can be synchronized using the GPIO pins on the serializer and deserializer. For the MAX9295A serializer used on the ADK, one must correctly set MFP7 pin for D12 disabled. This includes doing a register write of 0x07 to register 0x0007 (It is originally set to 0xF7). This correctly works with the priority established by Maxim to enable GP0 over D12.

For GMSL 1, the deserializer will need to have an input into the GPI pin with a pulse width of >0.35us. The GPI pin on the deserializer will need to be linked to MFP7 on the serializer.

To establish sync with the Boson the Boson needs to be configured to One Shot mode and External Sync Slave mode. Note that sync pulses must be applied before the Boson is configured to External Sync Slave mode or the Boson will fault and subsequently restart. The Boson looks at the rising edge of the incoming sync signal, and pulse width has a wide range of accepted variability. For the GMSL deserializer the sync pulse must be a 1.8 V logic high

pulse. Please see the Boson External Sync App Note Document for more information on the functionality of the sync feature with the Boson camera.

5 ADK Latency

5.1 Microbolometer Thermal Time Constant

The Boson Microbolometer sensor absorbs infrared radiation on the surface of the pixel, and continuously changes its temperature in relation to it. However, the change in temperature is not instantaneous. Instead, every microbolometer has a characteristic thermal time constant which dictates how quickly it responds to a change in irradiance.

In the case of the ADK, the nominal Thermal time constant is 8msec. This means that if the incoming radiation changed by a certain amount, it would take 8msec for the pixel to change (1-e-1) or 63.2% of the difference between its value prior to the change and its eventual steady-state value after the change. This effect is illustrated in Figure 4. Since the camera operates at 60 Hz, each frame period is just over 2 time constants, meaning that the thermal time constant is not noticeable in many environments.

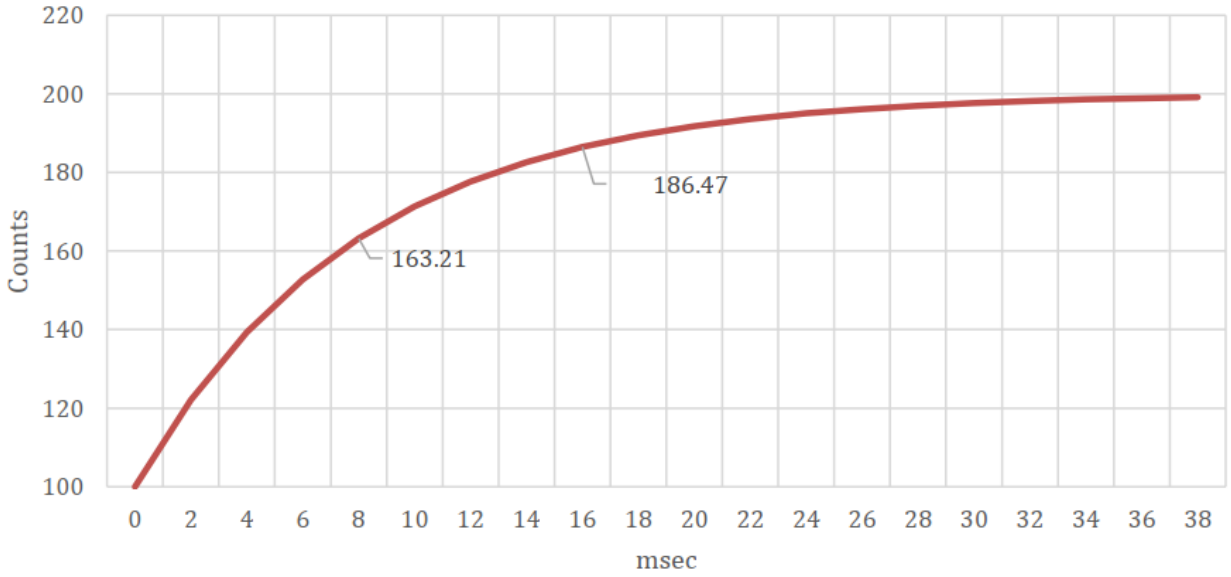


Figure 4. Pixel Response to a 100 Count Increase in Flux

It is worthy to note that the thermal time constant does add an extra latency to the video pipeline. Usually, the difference from one frame to another is not very abrupt. Smaller differences in LWIR radiation between frames hide the thermal time constant as the differences get smaller. Edges of fast-moving objects will have the most significant effect, creating a slight blur.

5.2 Progressive Video Output

The Boson thermal camera has a rolling shutter effect that is always present, but not always noticeable. This Boson is not a "snapshot" or "global shutter" camera. Each pixel is read individually at a slightly different time than the pixel to the left or right of it. Since the camera samples the top left pixel first, and the readout is row by row, an object that is moving very quickly may appear distorted in the image output. Approximately 32µs separate each line. This is most evident when vertically oriented objects are moving rapidly horizontally.

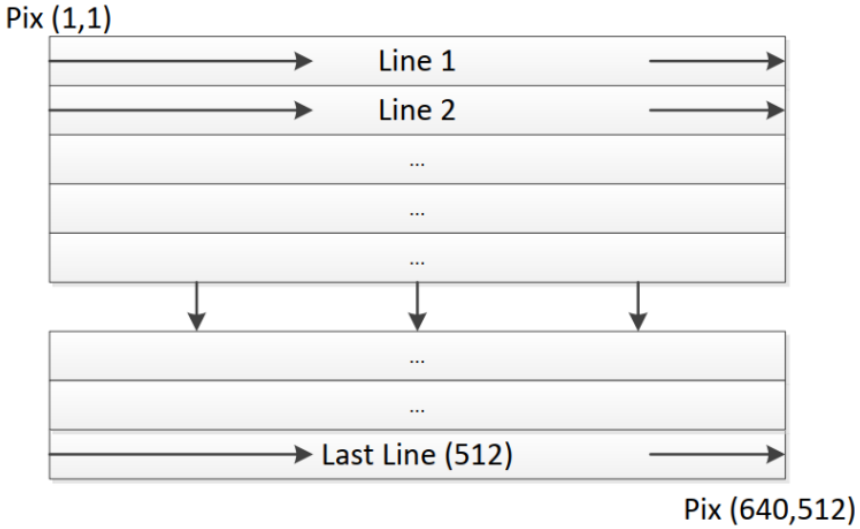


Figure 5. ADK Rolling Shutter

5.3 ADK Video Processing Pipeline

There is some latency in the FLIR ADK video pipeline. The camera reads the focal plane array (FPA), processes the image, and sends it to a parallel CMOS output. The personality board, providing either GMSL or USB output, is connected to the CMOS output. The delay from reading to first pixel from the FPA to when it is available on the CMOS output is 18ms. There will be additional delays if you are colorizing the image or using digital zoom.

Table 3: Timing of a frame from the ADK with average OFF.

Time (ms)	Event	Notes
0	Sync Pulse	Internal or External
0.3	ROIC reads first pixel	
16.7	ROIC read last pixel	16.4ms to read ROIC
18.3	CMOS output of first pixel	
35.0	CMOS output of last pixel	16.7ms to output frame

Table 4: Timing of a frame from the ADK with average ON.

Time (ms)	Event	Notes
0	Sync Pulse frame 2	Internal or External
0.3	ROIC reads first pixel frame 1	
16.7	ROIC read last pixel frame 1	16.4ms to read ROIC
16.7	Sync Pulse frame 2	
17.0	ROIC reads first pixel frame2	
33.4	ROIC reads last pixel of frame 2	
35.0	CMOS output of first pixel	
51.7	CMOS output of last pixel	16.7ms to output frame

When the averager is on, two frames of data from the ROIC are used to reduce noise. However if the change in pixel value is more than expected by noise, the averager assumes this is a change in the scene and uses the value from the more recent frame. This means for dynamic scenes, the latency is the time between ROIC reads first pixel and CMOS output of first pixel, or 18ms.

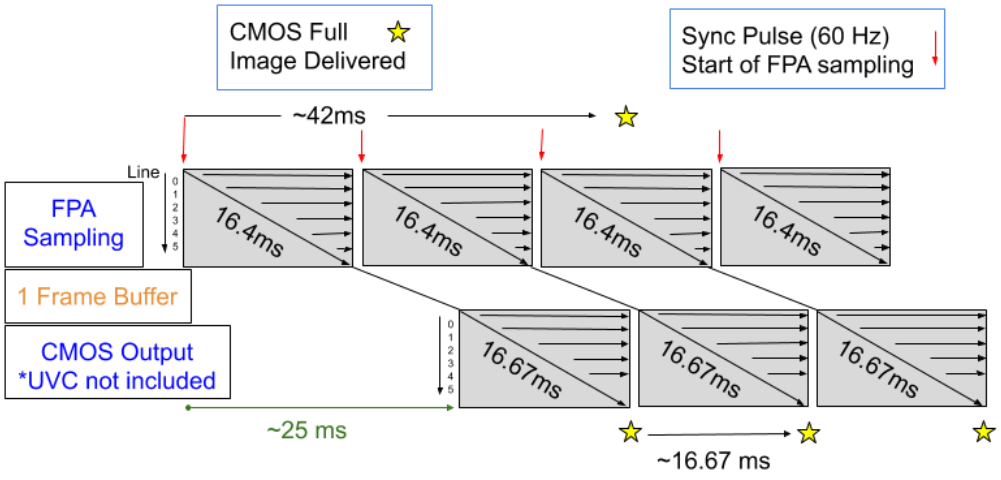


Figure 3: Output 60 Hz – averager OFF

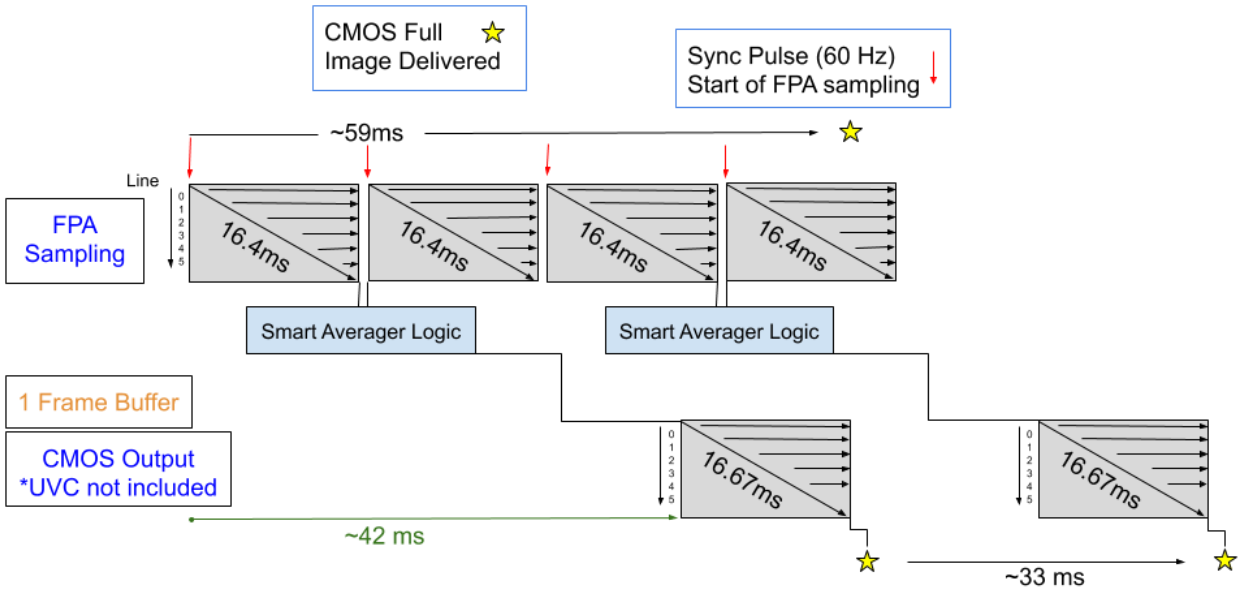


Figure 4Output 30Hz – averager ON

5.4 USB latency

The delay added by USB is not deterministic. The delays can change based on the topology and utilization of the USB network. These parameters are outside the FLIR's control and are not modeled.

5.5 GMSL latency

GMSL transmission time is negligible compared to the internal imaging pipeline. The serialize buffers one line of pixels before transmission. This buffering is negligible compared to the ADK's image processing.

5.6 Ethernet latency

The Ethernet Break Out Box (BoB) can provide a PTP (IEEE 1588) time stamp with each frame. The Ethernet BoB needs to be configured to enable PTP (GevIEEE1588) and set to slave only mode (GevIEEE1588Mode). On a Linux host with ptp4l installed, start PTP with "`sudo ptp4l -i enp0s25 -m`" where enp0x25 is the name of your Ethernet adapter. The timestamp on the image is approximately 18.4ms after the action in the scene.

6 Intrinsic Camera Calibration

Similar to a visible camera, a black and white chessboard pattern can be used with OpenCV to do intrinsic calibration to correct for effects to the lens. Because thermal cameras sees heat, more than just a chessboard is needed.

6.1 Building a Calibration Target

Print a checkboard pattern such as:

http://martinperis.blogspot.com/2011/01/opencv-stereo-camera-calibration.html

Mount the checkboard on a stiff material (such as cardboard) to keep the checkerboard flat. Then illuminate with the sun or a lamp that emits IR. The light should get hot. (LED or fluorescent lights will not work). For example:

https://www.amazon.com/HDX-250-Watt-Halogen-Portable-Light/dp/B06XH49LZB/

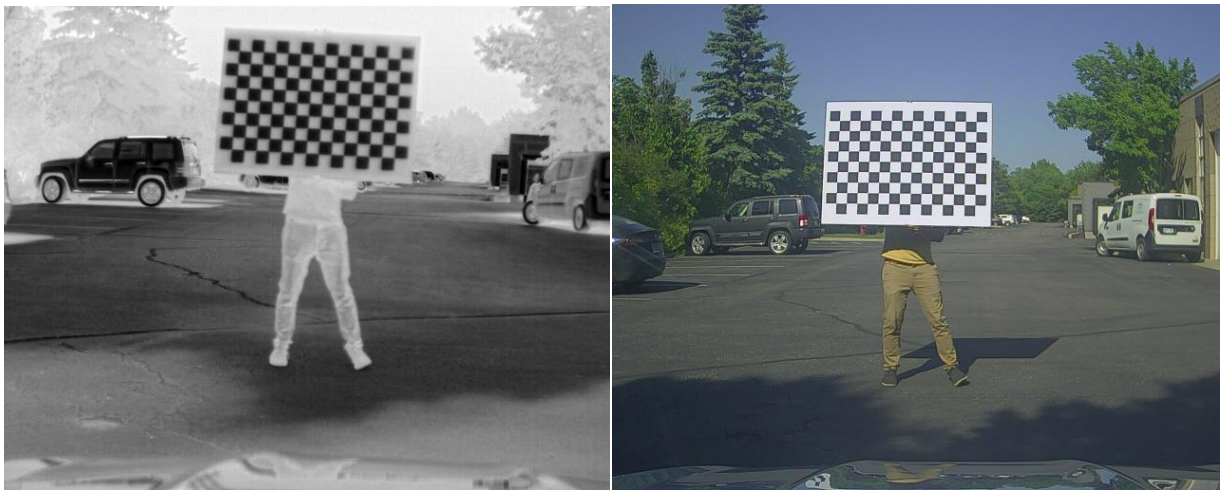


Figure 5: Thermal and RGB images taken by illuminating the checkerboard with the sun

When planning the size of your target there are a few things to keep in mind:

1) ADK cameras have their focus set at infinity. This means the target needs to be several feet from the camera to be in focus.
2) The target will need to be large enough in the FOV to clearly see the chessboard
3) The higher the contrast, the easier it is for the algorithm to find the chessboard. Keeping extra hot and cold objects out of the picture will help.
4) You may want to print a black box around you checkerboard.
5) During the calibration procedure, the orientation of the chessboard will need to move in relation to the camera.

6.2 Running OpenCV algorithm

Most of the open source intrinsic camera calibrations eventually call the OpenCV function findChessboardCorners. This function expects a white boarder around the chessboard. However, when imaged with the ADK, hotter is brighter and colder in darker. When illuminated with the light, the black squares absorb more radiation and heat up. So the dark squares in the visible image end up being the white squares in the thermal image and visa versa. The result the standard chessboard ends up with a black boarder and the grid is not recognized.

There are several ways of solving the 'findChessboardCorners needs a white boarder issue:

1. Print the chessboard with a black boarder. (This might cause problems for visible camera calibration
2. Put the camera into 'Black Hot' mode. Normally the hotter something is the lighter in appears in the image (aka White Hot). In Black Hot, hotter things are darker. Since the typical chessboard has a white boarder, it stays cooler and therefor lighting in this mode
3. In software, invert the image to make the boarder lighter (shown below)

Most software packages call findChessboardCorners with some options enabled. Some of those option make detecting the chessboard in thermal difficult. We recommend use the only the Adaptive Threshold option when calling findChessboardCorneres.

The following example code is based the the Camera Calibration library in the Image Pipeline ROS package. The code snippets show the lines of code changed in ROS/image_pipline/camera_calibration/calibrator.oy to calibrate a thermal cameras

1) Invert the image creating the equivalent of a photographic negative. Here I've added '255 – mono8a' to the last else statement.

```
  def mkgray(self, msg):
      """
      Convert a message into a 8-bit 1 channel monochrome OpenCV image
      """
      # as cv_bridge automatically scales, we need to remove that behavior
      # TODO: get a Python API in cv_bridge to check for the image depth.
      if self.br.encoding_to_dtype_with_channels(msg.encoding)[0] in
['uint16', 'int16']:
          mono16 = self.br.imgmsg_to_cv2(msg, '16UC1')
          mono8 = numpy.array(mono16 / 256, dtype=numpy.uint8)
          return mono8
      elif 'FC1' in msg.encoding:
          # floating point image handling
          img = self.br.imgmsg_to_cv2(msg, "passthrough")
          _, max_val, _, _ = cv2.minMaxLoc(img)
          if max_val > 0:
              scale = 255.0 / max_val
                  mono_img = (img * scale).astype(numpy.uint8)
          else:
```

```
              mono_img = img.astype(numpy.uint8)
              return mono_img
      else:
-         return self.br.imgmsg_to_cv2(msg, "mono8")
+         mono8a = self.br.imgmsg_to_cv2(msg, "mono8")
+         mono8b = numpy.array(255 - mono8a, dtype=numpy.uint8)
+         return mono8b
```

2) When calling findChessboardCorners, only use the adaptive threshold flag.

```
  def _get_corners(img, board, refine = True, checkerboard_flags=0):
    """
    Get corners for a particular chessboard for an image
    """
    h = img.shape[0]
    w = img.shape[1]
    if len(img.shape) == 3 and img.shape[2] == 3:
        mono = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    else:
        mono = img
    (ok, corners) = cv2.findChessboardCorners(mono, (board.n_cols,
-             board.n_rows), flags = cv2.CALIB_CB_ADAPTIVE_THRESH |
-             cv2.CALIB_CB_NORMALIZE_IMAGE | checkerboard_flags)
+             board.n_rows), flags = cv2.CALIB_CB_ADAPTIVE_THRESH)
    if not ok:
        return (ok, corners)
```