Table of contents Rev.23.04.2020/10:00h

Overview

This document describes how a software developer can access, control, and process the streaming infrared data from Fluke TV4x series thermal imagers. The techniques used rely on the *GigE Vision* protocol server built into these imagers and this document describes the Fluke-specific capabilities; it is the responsibility of the software developer to separately gain familiarity with the *GigE Vision* and *Gen<i>Cam* protocols.

Camera Setup and Configuration

2.1 Connecting to Web Configuration Tool

2.1.1 Web Configuration Tool

The web configuration tool allows for the configuration of the camera out of the box. It is supported for current versions of the Chrome web browser. Mozilla Firefox, Microsoft Edge, Opera, and Safari browsers as well. To navigate to the to the web configuration tool us one of the following methods:

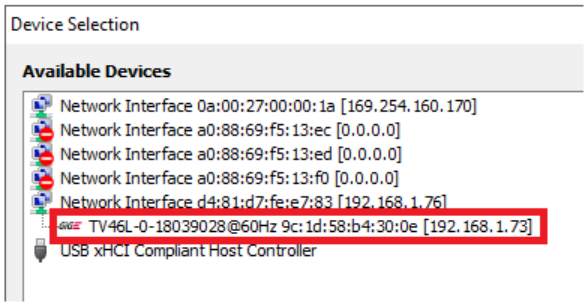
- Navigate to the Dynamic IP determined in section 2.1.1
  - Example: http://[ip_address]
- Navigate to the Imager Serial number located on the back of the imager
  - Example: http://[camera_serial_number].local
  - NOTE: This option only exists for Operating Systems that support Multicast-DNS (mDNS).
    - Windows 10 1511 (10586) and above
    - Mac OS X
    - Various Linux distros

2.1.2 Getting the Dynamically assigned IP address

Fluke TV4x Thermal Imagers come out of the box with dynamically assigned IP addresses. To get an IP address assigned, it is required to connect to a DHCP server, in most cases this would be a typical router or PC.

The dynamically assigned IP address can be found in the following ways:

- Router control panel
- Bonjour browser
- Many *GigE Vision* applications provide a device selection screen, seen in figure # below is Pleora's eBUS player.

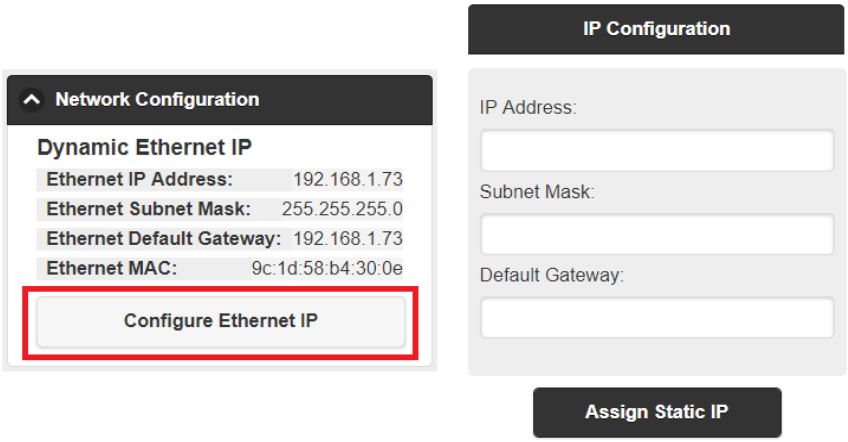


2.2 Using Web Configuration Tool

2.2.1 Setting a Static IP address

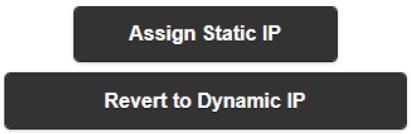
Setting a static (fixed) IP address on the Imager can be achieved by connecting to a DHCP server and navigating in a web browser to the web configuration tool.

In the Network Configuration section of the web configuration tool the button Configure Ethernet IP will bring up a popup that allows the setting of Static IP. Once the IP information has been filled out clicking Assign Static IP will validate and then set the static IP in the camera. (see figure # below) On a restart of the camera the new static IP will be active. It is a good idea to record the new static IP as it can be difficult to discover a device with a static IP where that IP is not known.



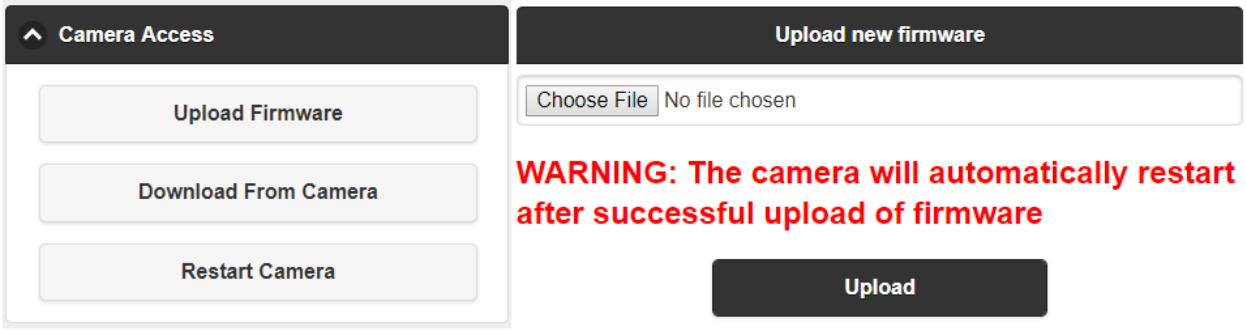
2.2.2 Reverting to Dynamically assigned IP address

The static IP can be reverted to a dynamically assigned IP address. This can be done on the web configuration tool under the Network Configuration section by clicking the Configure Ethernet IP button and then clicking the Revert to Dynamic IP from the IP Configuration popup menu. (see figure # below)



2.2.3 Updating Camera Firmware

A firmware .zip package can be uploaded in the web configuration tool by expanding the Camera Access panel and clicking the Upload Firmware button. After the Upload new firmware popup comes up click choose file, select the desired file from disk, and click the Upload button.



The most current camera firmware can be found here. ::LINK::

GigE Vision

3.1 Protocol

GigE Vision is a standard protocol for interfacing with cameras typically using UDP communications. One common media used for carrying the UDP communications is gigabit ethernet, but this is not required. Some Fluke cameras communicate using *GigE Vision* protocols over a RNDIS-enabled USB connection, although this has limits for range and communications flexibility. Gigabit Ethernet is preferred for best performance and installation convenience.

The *GigE Vision* standard is available here.

The *GenICam* Standard is available here.

The *GenICam* Standard Features Naming Convention is available here.

3.2 Connecting to the Camera

Standard *GigE Vision* 2.0 protocol can be used to discover the device and attempt to gain control of the device. Upon receiving a GigE Vision broadcast message, the Fluke imagers will reply with the standard GigE Vision discovery acknowledgement. Some notable fields to inspect are:

- Vendor: This field will be Fluke Process Instruments for the TV4x series
- Model: This field will contain the camera serial number followed an "@" symbol and the streaming data rate the camera is calibrated for
- Version: This field contains the firmware version. It is not possible to upload new firmware through GigE Vision on these cameras. To update to the latest firmware follow the Updating Camera Firmware section 2.2.3
- Device Class: These Fluke cameras are only GigE Vision Stream Protocol (GVSP) Transmitters.
- Serial Number: This field is the serial number of the Infrared Engine in the camera
- IP: This field is important if it is required to find the dynamically assigned or static IP address.

The *GenICam* XML file can be found using either URL1 at 0x0200 or URL2 at 0x0400, as they both return the single location where the XML is stored. This *GenICam* file can be parsed according to *GenICam* Standard Version 1.0 and Standard Feature Naming Convention Version 1.1.1. Further information about custom registers can be found in the next section on Configuring the Camera.

Once connected to the camera, a regular heartbeat command needs to be sent over the GigE Vision Control Protocol to keep the t

3.3 Configuring the Camera

Fluke has custom *GenICam* features for the Device Control category. There are two custom categories which contain the custom features; they are FLK_TI_INFO (section 3.3.1) and FLK_TI_ControlFeatures (section 3.3.2). There is also a custom feature under the ImageFormatControl category called the FLK_TI_StreamDataSourceSelector (section 3.3.3).

3.3.1 FLK_TI_INFO

The Table # below shows the useful custom features defined in this category as well as a description on how to interpret the results.

Register Name	Description
FLK_TI_Info_VLDataSize	Size of Visual Light image being sent *0xWWWWHHHH* *Most significant 16bits is width* *Least significant 16bits is height*
FLK_TI_Info_REDataSize	Size of Infrared image being sent

	0xWWWWHHHH *Most significant 16bits is width* *Least significant 16bits is height*
FLK_TI_Info_REDataRate	Frame rate of IR data
FLK_TI_Info_CurrentDeviceTemperatureC	Internal Temperature of device in Celsius
FLK_TI_Info_CriticalDeviceTemperatureC	Shutdown Internal Temperature of device in Celsius

3.3.1.1 FLK_TI_InfoSelector

The FLK_TI_InfoSelector feature is set to an enum value whose corresponding value can be read from the FLK_TI_InfoString. (default: FLK_TI_FirmwareVersion). An example would be if the FLK_TI_InfoSelector were set to FLK_TI_FirmwareVersion the firmware version would be made available to be read from the FLK_TI_InfoString, where the value would be held in the string (ex. “0.0.75”).

FLK_TI_InfoSelector (enum)	**FLK_TI_InfoString (value)**
FLK_TI_FirmwareVersion	Current firmware version on camera
FLK_TI_OSVersion	Current Linux version on camera
FLK_TI_PlatformId	SOC Platform Id
FLK_TI_RE_PlatformId	Engine Platform Id
FLK_TI_RE_DataRate	Frame rate of IR data

3.3.1.2 FLK_TI_Info_CalibrationRangeInfoQuery

The FLK_TI_Info_CalibrationRangeInfoQuery register has an EnumEntry for each calibration range. The number of calibration ranges can vary depending on the series and model of the camera. When the value of the EnumEntry is written to the FLK_TI_Info_CalibrationRangeInfoQuery the associate range will be set in the camera. Setting this range will change the range of values returned in a frame of streamed data. This means that a given power value pixel, which is represented by a 16-bit unsigned integer, will represent a temperature value from the lower boundary to the upper boundary. For a range with a lower boundary of -20^{o} C and upper boundary of 80^{o} C the power value of 0 will be the temperature -20^{o} C and the power value of 65535 (max uint16 value) will be the temperature 80^{o} C. The values in between these points follow the curves found in the calibration info discussed in section 4.2 Parsing the Calibration Info.

FLK_TI_Info_CalibrationRangeInfoQuery has corresponding boundary values which can be read from FLK_TI_Info_CalibrationRangeInfo_RangeSpan_LowerBoundary for the low end of the range , and from FLK_TI_Info_CalibrationRangeInfo_RangeSpan_UpperBoundary for the high end of the range. These values will always be in Celsius.

3.3.2 FLK_TI_ControlFeatures

3.3.2.1 Setting the focus

The focus for the Infrared Imager can be read and changed at any time while the device is being controlled by client. The focus distance is read and written in the units Millimeters. It can be read from the register FLK_TI_ControlFeature_CurrentFocusDistanceMm and set by writing to the FLK_TI_ControlFeature_SetFocusDistanceMm. It is important to note that the precision of focus motor cannot always match the same millimeter for the focus set. This means that if you set the distance to 1000 mm's the resultant focus could be 998 mm or 1001 mm. This would be the value that is read from the focus. The range for values that are allowed to be set are determined by the FLK_TI_ControlFeature_FocusDistanceMm_Max as the upper bound and FLK_TI_ControlFeature_FocusDistanceMm_Min as the lower bound. For the upper bound the value is typically much higher than the focus capabilities. Meaning that the if the focus distance is set to the upper bound (max) value, the position of the focus lens will reach its max much sooner. This will result in the value being read from the current focus distance register being much lower than that value. At the value of approximately 21 feet or 7000 mm's the focus lens will be at its infinite focus point beyond which will result in effectively no difference from max value.

Register	Value
FLK_TI_ControlFeature_SetFocusDistanceMm	Write the desired focus distance here, ensuring that the distance is in mm and within the bounds set by the **min** an **max**. When the command is performed the current distance register will update the distance.
FLK_TI_ControlFeature_CurrentFocusDistanceMm	Reading this value will give the **current** focus distance in mm.
FLK_TI_ControlFeature_FocusDistanceMm_Max	Reading this value will give the **max** value that should be used to set focus. (default: 1000000)
FLK_TI_ControlFeature_FocusDistanceMm_Min	Reading this value will give the **min** value that should be used to set focus. (default: 150)

3.3.2.2 FLK_TI_ControlFeature_REControlCmd

The FLK_TI_ControlFeature_REControlCmd register has the functionality to perform a variety of tasks depending on the enumerated value written to it. The enumerated values are listed in the table # below. The enumerated values all contain the prefix FLK_TI_ControlFeature which has been removed in the table for abbreviation. For example the enumerated value in the camera

FLK_TI_ControlFeature_REControlCmd_SelectCommandToExecute would be the value REControlCmd_SelectCommandToExecute in the below table.

Default: FLK_TI_ControlFeature_REControlCmd_SelectCommandToExecute, and no action performed while this value is set.

Value	Command
REControlCmd_SelectCommandToExecute	Default value, performs no action
REControlCmd_UseCalRange#	Switches the Calibration Range to the value denoted by #
REControlCmd_RequestFineOffset	Puts in a request to perform a fine offset (NUC) operation when IR engine is free.
REControlCmd_ExecuteFineOffset	Executes a fine offset.
REControlCmd_EnableAutomaticFineOffsets	Fine offsets will fire as the engine determines they are necessary, this is the default configuration.
REControlCmd_DisableAutomaticFindOffsets	Fine offsets will only fire as commanded. This will deteriorate the quality of the IR image over time.
REControlCmd_OpenREShutter	Open the NUC/FineOffset shutter
REControlCmd_CloseREShutter	Close the NUC/FineOffset shutter
REDataResolutionControlCmd_ReduceREDataResolutionBy2	If camera is 640x480 this will reduce the image to 320x240, this will reduce the data rate for streaming by 4, and can help with low bandwidth connections
REDataResolutionControlCmd_RestoreNativeREDataResolution	If a 640x480 camera has been set to 320x240, this will restore the original 640x480 resolution and increase the data transfer by 4.

3.3.3 FLK_TI_StreamDataSourceSelector

The value set in the FLK_TI_StreamDataSourceSelector register change what form the data is sent in, see table # below. These values can be changed at any time and the data type being streamed will be switched immediately. For the Combined_IR_VL_Data data type it is important to know the size of the visible light and infrared data, which can be found in the FLK_TI_Info_VLDataSize and FLK_TI_Info_REDataSize respectively (more details in section 3.3.1), so that it is known how much of the single frame should be processed as visible light and how much processed as infrared data.

(Default: VL_Data)

Value	Data Sending
VL_Data	Sending Visible Light images only, in the pixel format defined by the PixelFormat register, which at this time is only going to be YUV422
IR_Data	Sending Infrared images only, the pixel format is just raw uint16 power values. The values represent the entire range of values. Typically, the values returned are in a narrow band and if palettized will not provide significant contrast.
Interleaved_IR_VL_Data	This will send one frame of VL data followed by one frame of IR data, each being contained in its own block.
Combined_IR_VL_Data	This will send IR data and VL data as a single block with the infrared first followed by the visible light.
Blended_IR_VL_Data	This will send the IR data blended with the VL data blended together in fusion, and in the format of jpeg.

3.4 Streaming from the Camera

3.4.1 Packet Size Configuration

The Packet Size parameter in the *GigE Vision* should be set to 1444 or less. Even if jumbo packets are allowed on client-side network adapter the camera GigE Vision server only sends at rates of 1444 or less.

3.4.2 Selecting Stream Type

On initial connection to the FLK_TI_StreamDataSourceSelector, which determines data type coming across GVSP, will be set to Visible Light Data and will be sent in YUV422 Pixel Format. This can be switched during streaming or before streaming by writing the desired enum value to the FLK_TI_StreamDataSourceSelector register (more info in section 3.3.3).

3.4.3 Firewall

It is common for firewalls to block incoming UDP packets from an IP that has not been sent a packet. Sending a test packet on the GVSP before setting the acquisition start bit, will often resolve this issue. Reviewing the systems incoming UDP firewall configurations before streaming is good practice.

3.4.4 Start Streaming

Fluke cameras begin streaming as per the GigE Vision protocol by writing a 1 to the acquisition start register. The client must piece together the packets to form the entire frame where the data can then be processed accordingly. The next section (section 3.4.5) will discuss in further detail how the

Infrared (IR) is handled, although for the Visible Light (VL) displaying the YUV422 is standard. If you are new to YUV, consider the following resource: https://en.wikipedia.org/wiki/YUV

3.4.5 Processing the IR data

When sending Infrared images, the pixel format is just raw uint16 power values. The values represent the entire range of values (-20C–80C or -20C–1200C). Typically, the values returned are in a narrow band and if palettized will not provide significant contrast. Contrast can be provided by clamping the min and max value for a given frame. Additional adjustments can be made to the span to better palettize the data.

3.4.5.1 Reading Frame Headers on IR Data

For each frame received, a value exists embedded in the low order bit of the first uint16 values that will accumulate to the full header, as per the table # found below. There is also an example after that table that shows a possible coding solution; this will provide some further clarity into how the header values can be pulled out of the IR data.

Info Pixels/Bits in Order

Description	Values
Number of Info Bits which follows 8 pixels/bits	0 – Read 1 bit follows 1 – Read 2 bits follow 2 – Read 3 bits follow ... 255 - Read 256 bits follow
RE Data Resolution 2 pixels/bits	0 – RE data at native resolution 1 – RE data has dimensions of native resolution divided by 2
Calibration Range 4 pixels/bits	0 – First Cal. range 1 – Second Cal. range 2 – Third Cal. range ... and so on
Automatic Calibration Range Switching Flag 1 pixels/bits	0 - Automatic Cal. Range Switching Disabled 1 - Automatic Cal. Range Switching Enabled
Automatic Fine Offsets Enabled 1 pixels/bits	0 – disabled 1 – enabled
Automatic Fine Offsets Pending 1 pixels/bits	0 – not pending 1 – pending
NUC Shutter State 2 pixels/bits	1 – Obstruct Ir 2 – Unobstruct Ir
Encoded PIP Mode Ratio 3 pixels/bits	0: PIP mode has to be disabled 1: PIP mode ratio is 0.75

Make VL Invisible Flag 1 pixels/bits	0 - Don't make VL invisible 1 - Make VL invisible

```
// starting with a byte[] frame
// open the frame in a format to read uint16's out
using (MemoryStream stream = new MemoryStream(frame)){
using (BinaryReader reader = new BinaryReader(stream)){

//
byte numberOfPixelsToRead = 0;
for (int t = 0; t < 8; t++)
{
    // ( uint16 & 1 ) reads the lowest order bit out
    // ( << (7 - t) ) shifts the bit to the correct position, highest to lowest
    numberOfPixelsToRead += (byte)((reader.ReadUInt16() & 1) << (7 - t));
}

// create and fill an accumulator with the low order bits of following uint16 reads
List<byte> lobs = new List<UInt16>();
int index = 0;
while (index <= numberOfPixelsToRead)
{
    lobs.Add((byte)(reader.ReadUInt16() & 1));
    index++;
}

// read values out of the accumulator
int lob_index = 0;
byte REDataResolution = (byte)((lobs[lob_index++] << 1) + lobs[lob_index++]);
byte calibrationRange = (byte)((lobs[lob_index++] << 3) + (lobs[lob_index++] << 2) +
                               (lobs[lob_index++] << 1) + lobs[lob_index++]);
```

Getting Calibrated Temperature Values

4.1 Reading the Calibration Info

The custom FLK_TI_CalibrationInfo feature found in the custom FLK_TI_Info category in the GenICam descriptor contains the register to start reading the calibration info as a blob. There is a register which tells the size (in bytes) of the calibration data you will read. This is the FLK_TI_CalibrationInfo_SpanBytes_RegAddr register, which currently has a visibility of invisible so you will need to know the register value. The size reported via this register must a multiple of 4.

FLK_TI_CalibrationInfo_SpanBytes_RegAddr register: **0x0000A100**

The number of bytes to read from the FLK_TI_CalibrationInfo is the value returned from the 4 byte value read from the FLK_TI_CalibrationInfo_SpanBytes_RegAddr register.

4.2 Parsing the Calibration Info

After reading the Calibration Info into memory on the client side and validating the size it is important to also validate that the calibration info is for the appropriate engine. To confirm this, inspect the first 32-bit word and if it is equal to 0x52696d01 then you interpret the rest of the calibration info data as the structure(s) provided below:

```
struct CalRangeDescriptor
{
	float calRangeMinTempC;			///< Min calibrated temperature for range
	float calRangeMaxTempC;			///< Max calibrated temperature for range

	///< Min displayed temperature (may be less than calMinTemp)
	float displayRangeMinTempC;

	///< Max displayed temperature (may be more than calMaxTemp)
	float displayRangeMaxTempC;

	///< Min palette resolution for manual palette scaling mode
	float minManualPaletteSpanDegreesC;

	///< Min palette resolution for automatic palette scaling mode
	float minAutoPaletteSpanDegreesC;

	unsigned int numUInverseCurvesDescriptors;

	/* 11 uinverse curves each with 5 float values */
	/* 5 values are: u0, u1, u2, startTempC, endTempC */
	float UInverse[11][5];   ///< U-Inverse transfer function coefficients array
};

struct CalRangesData
{
	unsigned int magic;     /* 0x52696d01 */
	unsigned int numEnabledRanges;

	/* one bit per range (Least Significant Bit would be rangeA) */
	/* 0000 0000 0000 0000 0000 0000 0000 0011 would be rangeA and rangeB */
	/* this allows for a max of 32 ranges, although only 3 possible on this camera */
	unsigned int enabledRangesMask;

	/// calibrationDate is a single 32 bit word with all date information encoded
	union
	{
		unsigned int encoded;
		struct
		{
			unsigned short runNumber : 2;
			unsigned short day : 5;
			unsigned short month : 4;
			unsigned short year : 5;
			unsigned short unused;
		};
	} calibrationDate;

	CalRangeDescriptor calRangeDescriptors[3];
	unsigned int checksum;
};
```

Temperature Calculations

After reading the CalibrationInfo from the camera a power to temperature look up table can be built up and used to map the data coming through the IR_Data stream to the appropriate temperature values. The steps are as follow:

1. Cast the read Calibration Info into CalRangesData struct as described in section 4.2
2. Check numEnabledRanges to see how many calibrations ranges are available
    a. These cal ranges can be read from calRangeDescriptor
3. The CalRangeDescriptor struct is used to converted to a power to temperature look up table, to do so see the following example

Within the CalRangeDescriptor there are 11 Inverse segments, and each segment containing 5 values.

- float u0 = uinv[segment _index][0]
- float u1 = uinv[segment _index][1]
- float u2 = uinv[segment _index][2]
- float current_UInverse_StartTemp = uinv[segment_index][3]
- float current_UInverse_EndTemp = uinv[segment _index][4]

The Power to Temperature and Temperature to Power can be calculated by using these values. The look up table is built by computing each segment up to the value provided by numUInverseCurvesDescriptors.

To process a single UInverse segment, convert the start and stop temperature values to power values using the provided Temperature -> Power equation using the appropriate segment. Then iterate from power start value to power stop value for the segment using the Power to Temperature table below.

Once one segment is complete move on to the next segment and add the values for that segment until all the required (numUInverseCurvesDescriptors) segments have been added

Temperature Celsius (float) -> Power (uint16)

```
float dresult = (((float)temperatureValueCelsius * uinv.U2) + uinv.U1) * (float)temperatureValueCelsius
+ uinv.U0;

result = (Int32)Math.Round(dresult);
```

Power (uint16) -> Temperature Celsius (float)

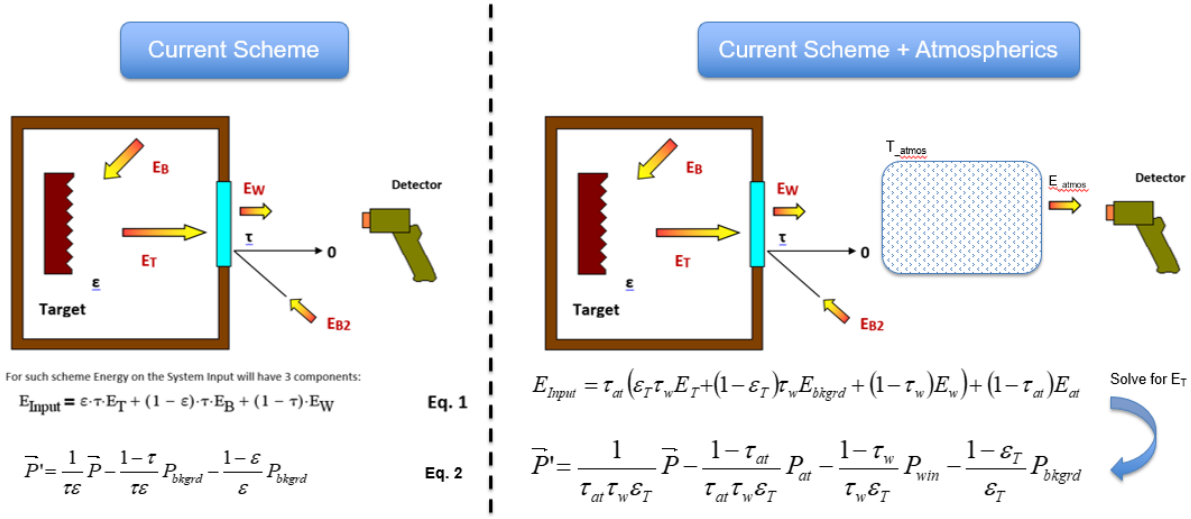
```
float u0 = uinv.U0;
float u1 = uinv.U1;
float u2 = uinv.U2;
float U1mulU1 = u1 * u1;
float u1_negative = -u1;
float U2mul2 = u2 * 2;
float U2mul4 = 4.0f * u2;
float U1mulU1_minus_U2mul4mulU0 = U1mulU1 - U2mul4 * u0;

float temperatureValue = (u1_negative + (float)Math.Sqrt(U1mulU1_minus_U2mul4mulU0 + U2mul4 *
(float)energyValue)) / U2mul2;
```

EBT & EBT + humidity compensation:

To get the most accurate power to temperature conversion use the equations below to find Power prime and use that value to plug into the previously calculated look up table.

- Adjust current Emissivity and Transmissivity scheme to include atmospherics as below



Specifically for humidity compensation:

- Simplify the atmospheric temperature to be same as target background
- Substitute Beer's equation for the atmospheric transmission
- User needs to input 2 additional values
  - Relative Humidity [RH = 0 – 99%]
  - Target Range [D = 0 - 5 km]

$$\vec{P}' = \frac{1}{\tau_{at}\tau_w\varepsilon_T}\vec{P} - \frac{1-\tau_{at}}{\tau_{at}\tau_w\varepsilon_T}P_{at} - \frac{1-\tau_w}{\tau_w\varepsilon_T}P_{win} - \frac{1-\varepsilon_T}{\varepsilon_T}P_{bkgrd}$$

Simplify to one background

$$\vec{P}' = \frac{1}{\tau_{at}\tau_w\varepsilon_T}\vec{P} - \frac{1-\tau_{at}}{\tau_{at}\tau_w\varepsilon_T}P_{bkgrd} - \frac{1-\tau_w}{\tau_w\varepsilon_T}P_{bkgrd} - \frac{1-\varepsilon_T}{\varepsilon_T}P_{bkgrd}$$

$$\tau_{at} = e^{(-\sigma_{ext}R)}$$

Beer's Law

Use LUT for this

Smooth LUT	
Relative Humidity (%)	Extinction Coefficient (km^{-1})
1.0000	0.0090
5.0833	0.0090
9.1667	0.0091
13.2500	0.0092
17.3333	0.0094
21.4167	0.0096
25.5000	0.0099
29.5833	0.0102
33.6667	0.0106
37.7500	0.0111
41.8333	0.0116
45.9167	0.0122
50.0000	0.0128
54.0833	0.0136
58.1667	0.0144
62.2500	0.0152
66.3333	0.0162
70.4167	0.0172
74.5000	0.0183
78.5833	0.0196
82.6667	0.0213
86.7500	0.0254
90.8333	0.0329
94.9167	0.0447
99.0000	0.1000