SDK User Guide

Seek Thermal Incorporated

Version 4.4.2.20

Table of Contents

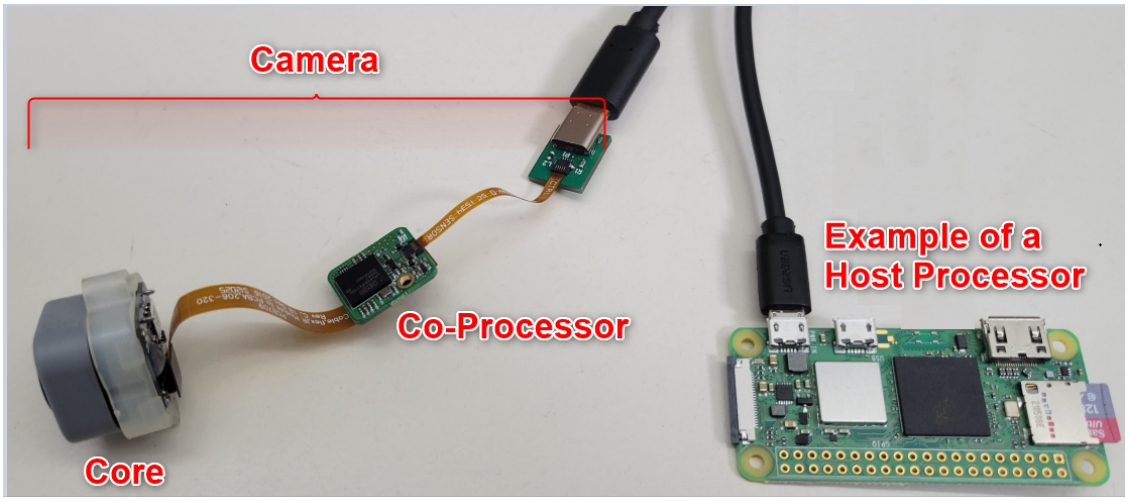
Introduction

The Seek Thermal SDK 4.x allows developers to interface with our thermal cameras easily and efficiently. It uses an event-driven architecture that allows for responsive interaction and control. In addition to the event-driven architecture, the SDK is designed with modularity and separation of responsibility in mind. An important realization of the new design is easy use of multiple cameras for the first time ever.

This document covers the general concepts on how to use the SDK. For details about specific APIs, please refer to the SDK C Programming Guide.

Terminology

Seek uses the terms described below for their system components. The picture shows an example of a full system with a Raspberry Pi Zero 2 W. The SDK supports many different architectures to allow customers to pick a host operating system and processor that best suits their needs. A full system tpycailly contains the following components:



- **Host Processor**: Processor that runs the Seek Thermal SDK along with the users application.
- **Camera**: Generic term for the core, flex cables, co-processor, etc.
- **Core**: The sensor, lens, housing, shutter (for Mosaic), etc. Sometimes called the camera or thermal camera also.
- **Co-processor**: The required co-processor that allows the core to interface to the host processor.

High-level Architecture

The main purpuse of the Seek Thermal SDK is to provide frames to the application. There are three different groups of frames (thermography, image, and partially processed). The application must perfom some simple steps to start receiving frames.

The application must perform three actions with the SDK

1. Creating a SDK camera manager
2. Starting a camera session when a camera is plugged in
3. Handling the frames as they come in

The SDK is "event-driven". This means the application does not poll to see if a camera is attached or if frames are ready. Instead the application sets up callbacks (event and frame) that are called by the SDK accordingly. So the event callback is called when a camera is plugged in (or unplugged). Similarly, the frame callback is called when the SDK receives a new frame from a camera.

Below is a psuedo-code template (no error checking) for the above three main steps. There are other APIs in the SDK, but this shows the bare minimum that can be used. Please use this code snippet as a reference as we'll dive into each section more afterwards. Please note the below template is very similar to the actual code in the `seekcamera-sdl` and `seekcamera-simple` examples.

```
/***********************************************************************/
/* #3: Process incoming frames                                         */
/***********************************************************************/
void handle_camera_frame_available(seekcamera_t* camera, seekcamera_frame_t*
camera_frame, void* user_data)
{
    (void)camera;
    auto* renderer = (seekrenderer_t*)user_data;
    // This mutex is used to serialize access to the frame member of the renderer.
    // In the future the single frame member should be replaced with something like a
FIFO.
    std::lock_guard<std::mutex> guard(renderer->the_mutex);

    // Lock the seekcamera frame for safe use outside of this callback.
    seekcamera_frame_lock(camera_frame);
    renderer->is_dirty.store(true);

    // This mutex is used to serialize access to the frame member of the renderer.
    // In the future the single frame member should be replaced with something like a
FIFO.
    renderer->frame = camera_frame;
    g_is_dirty.store(true);
    g_condition_variable.notify_one();
}

/***********************************************************************/
/* #2: Handle when asynchronous camera events                          */
```

```
/*       * plugged-in when the application starts                    */
/*       * plugged-in while the application is running               */
/*       * unplugged while the application is running                */
/*       * camera errors                                             */
/*********************************************************************/
void camera_event_callback(seekcamera_t* camera, seekcamera_manager_event_t event,
                           seekcamera_error_t event_status, void* user_data)
{
    switch(event)
    {
        case SEEKCAMERA_MANAGER_EVENT_CONNECT:
            handle_camera_connect(camera, event_status, user_data);
            break;
        case SEEKCAMERA_MANAGER_EVENT_DISCONNECT:
            handle_camera_disconnect(camera, event_status, user_data);
            break;
        case SEEKCAMERA_MANAGER_EVENT_ERROR:
            handle_camera_error(camera, event_status, user_data);
            break;
        case SEEKCAMERA_MANAGER_EVENT_READY_TO_PAIR:
            handle_camera_ready_to_pair(camera, event_status, user_data);
            break;
        default:
            break;
    }
}

/*********************************************************************/
/* #1: Starting a camera manager session                             */
/*********************************************************************/
int main()
{
    seekcamera_manager_t* manager = nullptr;
    seekcamera_error_t status = seekcamera_manager_create(&manager,
SEEKCAMERA_IO_TYPE_USB);
    if(status != SEEKCAMERA_SUCCESS)
    {
        std::cerr << "failed to create camera manager: " <<
seekcamera_error_get_str(status) << std::endl;
        return 1;
    }
}
```

Create Manager (#1)

The code snippet aboves shows the two main actions that are needed to create the manager.

- Call `seekcamera_manager_create()` to initialize the SDK's internal structures and tell it which type of peripherals to use. Currently, only `SEEKCAMERA_IO_TYPE_USB` is currently supported with the SDK.

- Once the manager is successfully initialized, the application needs to register an event callback via the `seekcamera_manager_register_event_callback()` API. In the above snippet, the name of this callback is `camera_event_callback()`. The `user_data` structure is opaque to the SDK. It is never touched by the SDK and is simply passed into the event callback.

The SDK also provides the `seekcamera_manager_destroy()` API to gracefully shutdown the manager. In the above code snippet, this is called after the application determines it is ready to terminate.

Starting Session (#2)

Please refer to the `camera_event_callback()` function in the code snippet above. This is the `seekcamera_manager_event_callback_t` function registered with the SDK in `main()`. This function is called by the SDK for the following reasons

- A camera was connected (plugged-in) when the camera manager was created
- A camera is connected (plugged-in) while the application is running
- A camera is disconnected (unplugged) while the application is running
- a camera has an internal error

Since the v4.x SDK supports multiple cameras, the first parameter of the event callback function is `seekcamera_t *camera`. This is a unique handle for each camera in the system.

The next parameter is the type of event (`seekcamera_manager_event_t`) that occurred. The next parameter is the event status and finally the user supplied data specified in the `seekcamera_manager_register_event_callback()` call. Again the `user_data1` element is not touched by the SDK and it's contents are completely up to the application.

Let's look at each type of event in the following sections.

Camera Connected

When a plugged in camera is detected, the connect event comes into the event callback. When this occurs, the application needs to register a frame callback and start a session for the new camera.

The application registers a frame callback to handle the asynchronous arrival of frames via the `seekcamera_register_frame_available_callback()` API. In the above code snippet, the callback is the `frame_available_callback()` function. Additionally the application can supply application specific data for that camera. That application specific data will be passed into the frame callback. It is opaque to the SDK and will not be modified by the SDK. In the above snippet, this is the `user_data2` variable. In the SDK's `seekcamera_simple` example, there is a camera context structure used for each unique camera. In this snippet it holds a pointer to a .csv file to hold the pixel thermography values for that camera.

After the frame callback is set, the application starts the session for that camera via the `seekcamera_capture_session_start()` API. To help performance of the SDK, the application must specify the types of frames for the SDK to process. For example, some applications only want imaging frames or only thermography frames, while some want both. More details on this topic are in the Frame Type Selection section.

Camera Disconnected

When is a camera is unplugged, the SDK will report a disconnect event in the event callback. The application should call the `seekcamera_capture_session_stop()` API to clean up the resource allocated for this camera. If the camera is plugged back in, the connect event will be generated again.

Camera Ready to Pair

The Micro Core cameras need to pair with the host processor running the SDK. Needed information from the camera is placed onto the host processor during the call to `seekcamera_store_calibration_data()`. Whenever a camera is connected to a new host processor, this process must be done. The SDK's `seekcamera_sdl` example shows an example of pairing.

The Mosaic Cores mechanism when attaching to a new host processor is different and does not require the explicit call to `seekcamera_store_calibration_data()`.

Camera Errors

Asynchronous errors can occur and are reported via the event callback function.

Handling Frames (#3)

When a frame from a camera is received by the SDK, the registered frame callback for that camera is called. For the above template, this is the `frame_available_callback()` function. Since there can be multiple cameras in the system, the `camera` and `user_data` fields can be used to help determine which camera has the new frame.

The `seekcamera_frame_t` type is an opaque type and the application should not directly access it. Instead it must use the SDK APIs to extract the needed information. For example

- `seekcamera_frame_get_frame_by_format()`: returns the desired frame_format.

Simiarly the `seekframe_t` type is opaque and should not be directly accessed. Please use the APIs in the SDK to extract the needed data. For example

- `seekframe_get_data()`: returns the actual frame buffer (e.g. 200x150 or 320x240 pixel array).
- `seekframe_get_header()`: returns the frame header (which has many interest fields like spot temperature, timestamps, etc.).

The frames and frame callback are discussed in more details in the Frames section.

Restarting

There is not a specific API for restarting a camera session. Instead, the application should simply call the `seekcamera_capture_session_stop()` API and then `seekcamera_capture_session_start()` when desired. The 'stop' must not be called in frame callback. Additionally, make sure no frames are locked when the 'stop' is called (refer to Frame Ownership for more details).

Frames

Seek Thermal uses a standardized data layout for all the frames formats. Frame data is stored in row-major order, meaning rows vary fastest and then columns. Pixel channels are stored contiguously, with optional padding between pixels.

A high-level flow graph of the Seek Thermal frame processing pipeline is shown below.

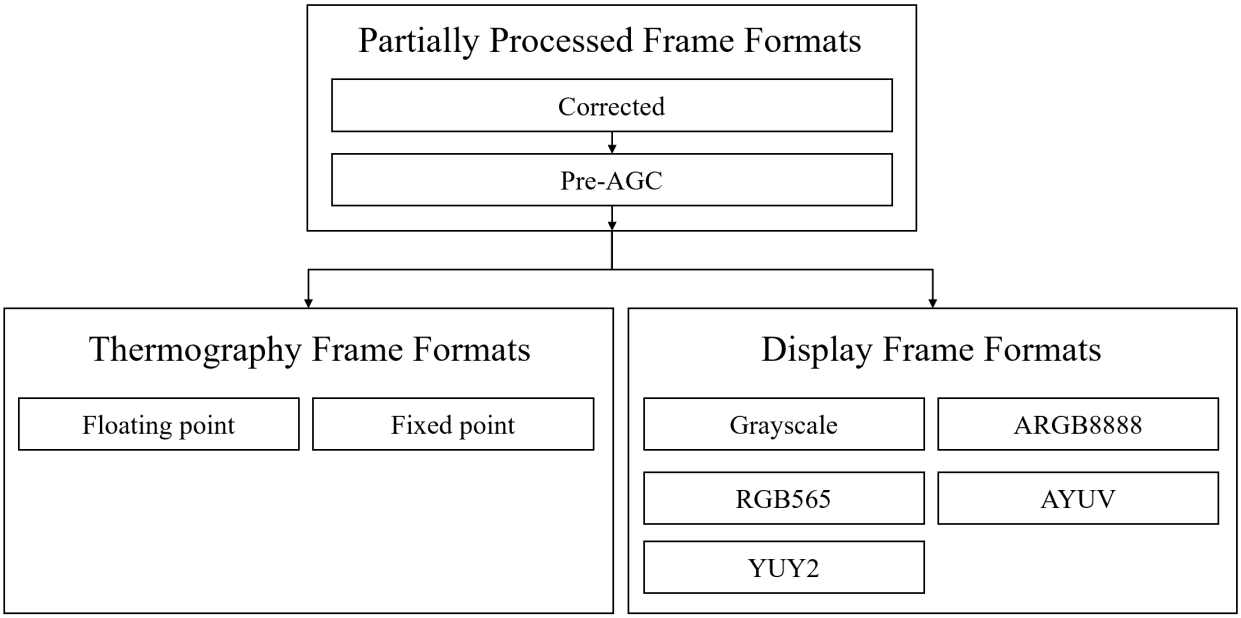


Figure 1. Frame processing flow graph

For a detailed description of each supported frame formats, please refer to the SDK C Programming Guide.

Frame Data Structure

There are four main data structures when dealing with frames.

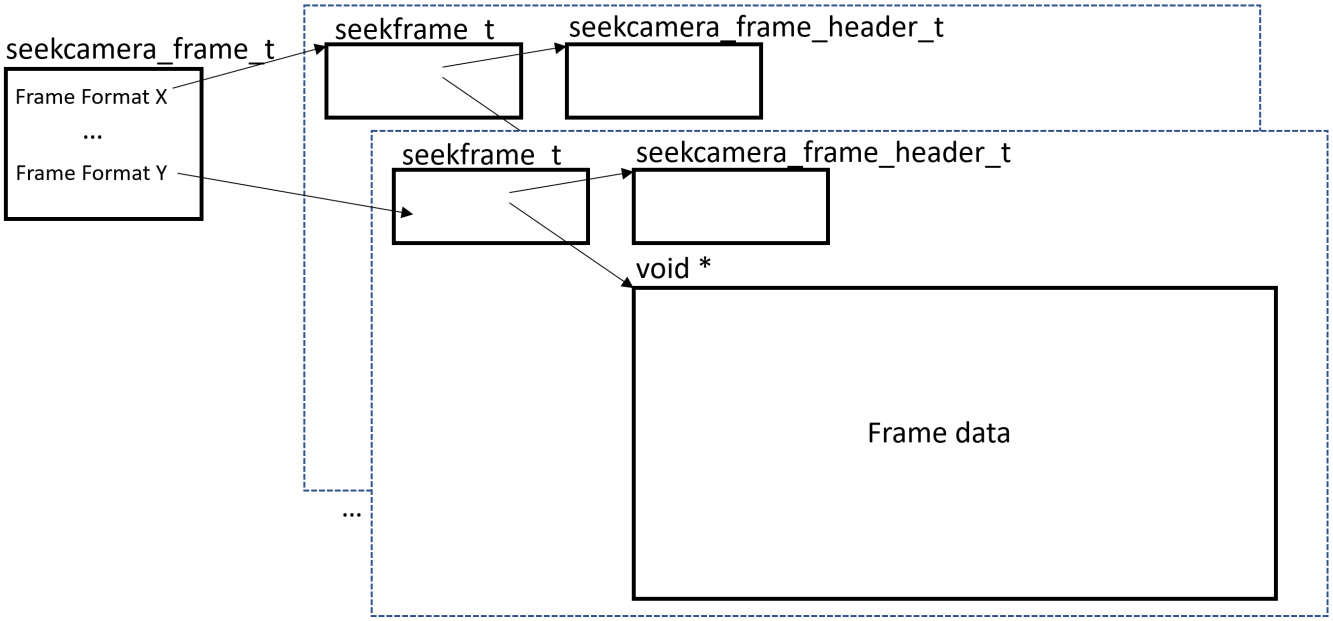


Figure 2. Frame Data Structures

- `seekcamera_frame_t`: This is a "meta" frame from a camera. It is passed into the frame callback. It is used to obtain the different frame formats (and related frame data and headers). This is an opaque structure that should not be directly accessed. Instead the application needs to use the APIs provided in the SDK.
- `seekframe_t`: This data structure contains the actual frame for a specific frame format. This is an opaque structure that should not be directly accessed. Instead the application needs to use the APIs provided in the SDK. This structure can be obtained from the `seekcamera_frame_t` structure via the `seekcamera_frame_get_frame_by_format()` API. Note there can be multiple `seekframe_t` in a `seekcamera_frame_t`. This is discussed in the next section.
- `seekcamera_frame_header_t`: Header information for a specific `seekframe_t` frame. The data structure is defined and can be directly accessed or there are many "helper" functions to retrieve fields from the structure. The header structure can be obtained via the `seekframe_get_header()` API.
- `void *` frame data: The actual frame data two dimensional array is a `void *` since the size of the elements depends on the frame formats (e.g. formats are either 8, 16 or 32 bits). There are several APIs in the SDK to extract the actual frame data from the `seekframe_t` opaque structure (e.g. `seekframe_get_data()`, `seekframe_get_row()`, and `seekframe_get_pixel()`).

Frame Format Selection

The SDK allows an application to receive multiple frame formats within the frame callback. For example, some applications only want thermography frames, while other applications might want both thermography and image frames.

When the camera session is started (via `seekcamera_capture_session_start()`), the second parameter to the API is the frame_format. The application must specify which frame formats are desired. If only one type is needed (e.g. the imaging `SEEKCAMERA_FRAME_FORMAT_COLOR_ARGB8888` one), simply pass in that `seekcamera_frame_format_t` value. If multiple frame formats are needed, 'add' in the desired formats. For example

```
frame_format = SEEKCAMERA_FRAME_FORMAT_THERMOGRAPHY_FLOAT |
               SEEKCAMERA_FRAME_FORMAT_COLOR_RGB565;
seekcamera_capture_session_start(camera, frame_format);
```

Then in the frame callback, specify which frame format(s) are needed.

```
seekcamera_frame_get_frame_by_format(camera_frame,
                                     SEEKCAMERA_FRAME_FORMAT_THERMOGRAPHY_FLOAT,
                                     &therm_frame);
//process thermography frame that contains 32-bit data

seekcamera_frame_get_frame_by_format(camera_frame,
                                     SEEKCAMERA_FRAME_FORMAT_COLOR_RGB565,
                                     &image_frame);
//process RGB565 frame that contains 16-bit data
```

The application can only specify at most one thermography, one partially processed, the greyscale imaging frame, and one additional color imaging frame format at a time. So you cannot get both `SEEKCAMERA_FRAME_FORMAT_CORRECTED` and `SEEKCAMERA_FRAME_FORMAT_PRE_AGC` in a session.

Frame Callback

Here is the type of the frame callback.

```
typedef void (*seekcamera_frame_available_callback_t)(
    seekcamera_t* camera,
    seekcamera_frame_t* frame,
    void* user_data);
```

The frame callback is written by the application and is responsible from processing incoming frames from the camera(s). As noted earlier, multiple cameras can be in the system. Each camera in the system will have a unique `seekcamera_t` pointer. This and/or the `user_data` parameter can be used to determine which camera has supplied the `seekcamera_frame_t` frame.

Generally the application processes the frame in the context of the frame callback. There are locking mechanism if the frame is queued up and processed outside of the frame callback.

The frame callback should not block waiting for other resources because this may back up the flow of the next incoming frame(s).

Frame Header Data

With each frame, there is a frame header of type `seekcamera_frame_header_t`. The header can be obtained via the `seekframe_get_header()` API. All of the fields are listed in the SDK C Programming Guide, but in general some of the fields are static for a camera (e.g. `serial_number`, `width`, and `type`). Other fields are more dynamic in natural (e.g. `thermography_spot_value` and `timestamp_utc_ns`).

Two fields that might be of interest are

- `timestamp_utc_ns`: This an a UTC timestamp in nanoseconds that can be used to synchronize multiple cameras. The timestamp is determined when the frame arrives into the host processor. The timestamp is applied by the SDK immediately after the last row of the frame is received on the host processor.
- `environment_temperature`: A non-calibrated estimate of the sensor's temperature. Even though you cannot get the exact tempetaure of the sensor, this paramter can be used to see if the sensor's temperature is changing.

Frame Data

Here are the APIs in the SDK to extract the actual frame data from the `seekframe_t` opaque structure

- `seekframe_get_data()`: returns the two dimensional array
- `seekframe_get_row()`: returns a single row of the two dimensional array
- `seekframe_get_pixel()`: returns a single pixel of the two dimensional array

The actual format of the frame data is dependent upon the frame format. Please refer to the SDK C Programming Guide for specific details on the layout of the data.

Frame Ownership / Locking

The application has complete ownership of the frame while in the frame callback. The SDK will not modify the frame while the application processes the frame in the callback. Upon exiting the frame callback, the SDK regains ownership and is free to re-use or modify the frame.

There are times where it might be benefical or more natural for the application to access/process an incoming frame outside the context of the frame callback. In this case the application can 'lock' the frame while in the frame callback. Later it can 'unlock' the frame once the processing is completed. If the frame is 'locked', the SDK will not modify the frame until the 'unlock' occurs even after the frame callback returns.

Calling the `seekcamera_frame_lock()` in the frame callback 'locks' the frame. After processing, the application must call `seekcamera_frame_unlock()` to 'unlock' the frame, which allows the SDK to re-use the frame. If the application fails to 'unlock' the frames in a timely manner, the SDK might eventually have a memory error because the SDK has a finite number of frame buffers.

Please note the `seekcamera-sdl` example in the SDK uses these locking mechanism since the SDL handling must be performed in `main()`.

Thermography Frames

The SDK provides thermography (radiometric) frames. It supports either `SEEKCAMERA_FRAME_FORMAT_THERMOGRAPHY_FLOAT` and `SEEKCAMERA_FRAME_FORMAT_THERMOGRAPHY_FIXED_10_6` frame formats (although you can only get one or the other for a camera during a session). Please refer to the SDK C Programming Guide on how to manage the `SEEKCAMERA_FRAME_FORMAT_THERMOGRAPHY_FIXED_10_6` data.

The SDK's `seekcamera-simple` shows an example of receiving and processing thermography frames. Here is an example of the .csv file generated by the `seekcamera-simple` example. Note the header information at the top of the file and temperature for every pixel in the camera.

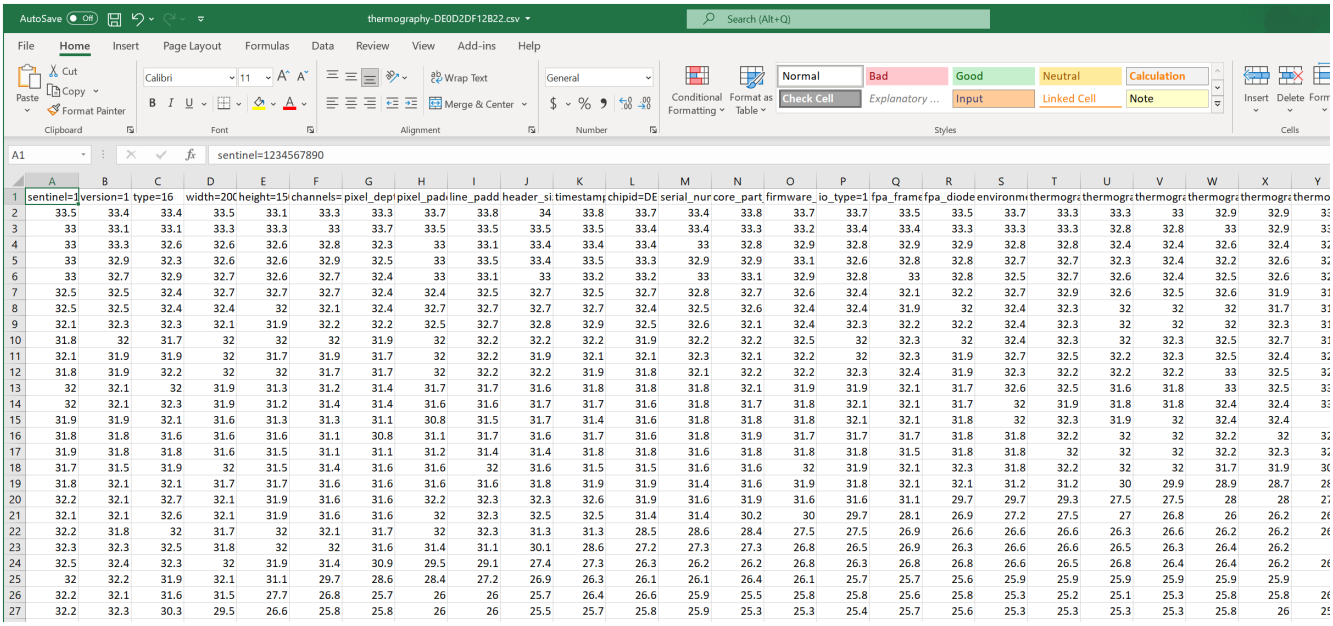


Figure 3. Thermography Frames from `seekcamera-simple`

The `SEEKCAMERA_FRAME_FORMAT_THERMOGRAPHY_FIXED_10_6` is useful for:

- **Smaller payload**: The pixels are 16-bits instead of 32-bit float values.
- **Reduced CPU usage**: If the processor does not have hardware floating point, the fixed frames do not incur as much floating point arithematic.
- **Custom AGC/color algorithms**: The fixed frames may be useful for advanced users writing their own AGC/color algorithms.

Window Size Controls

The `seekcamera_set_thermography_window()` API can be used to modify the default size of the thermography frames. By default every pixel in the thermography frames has a temperature. The `seekcamera_set_thermography_window()` API allows you to create a rectangle in the frame. Only that rectangle will have valid temperature values (every pixel outside the rectangle will have zero). This can be used to help reduce CPU usage.

Image Frames

On every image frame there is always a minimum and maximum temperature the scene. These values almost always change from frame to frame. The SDK has Automatic Gain Control (AGC) support and distributes the color palettes across the dynamically changing temperature range automatically. The following sections will dive into this rich topic of different imaging modes and the control mechanisms in the SDK.

The SDK's `seekcamera-sdl` shows an example of receiving and processing image frames.

Image Frame Formats

The SDK offers the several image frame formats. Which one the application uses depends primarily on how the image frame will actually be displayed. For example, the SDK `seekcamera-sdl` example uses the ARGB888 format because this format is expected by the SDL package. Additionally, since the formats have different sizes, one might be preferred due to smaller frame buffers.

Here are the different image frame formats in the SDK

Frame Format	Pixel Depth (bits)
`SEEKCAMERA_FRAME_FORMAT_GRAYSCALE`	8
`SEEKCAMERA_FRAME_FORMAT_COLOR_ARGB8888`	32
`SEEKCAMERA_FRAME_FORMAT_COLOR_RGB565`	16
`SEEKCAMERA_FRAME_FORMAT_COLOR_AYUV`	32
`SEEKCAMERA_FRAME_FORMAT_COLOR_YUY2`	16

Color Palettes

The SDK offers many imaging frame formats, display algorithms, and fine tuning APIs. First let's understand the color palettes and how they work. Afterwards we'll look into the actual imaging options in the SDK.

The color palette is spread across the image frames differently depending on the Automatic Gain Control (AGC) mode. This topic is discussed more in depth in the Linear and HistEq AGC Mode sections, but we'll look at the color palettes themselves first.

The SDK comes with a set of predefined color palettes. Which palette to use is entirely up to the application. Many customers allow the palettes to be changed dynamically in their final product since it is bit of a personal preference. Some of the palettes do offer a high contrast (e.g. Prism, Spectra, and Hi). The `seekcamera_set_color_palette()` API can be used to set the color palette to any of the following palettes.

```
typedef enum seekcamera_color_palette_t
{
    SEEKCAMERA_COLOR_PALETTE_WHITE_HOT = 0,
```

```
    SEEKCAMERA_COLOR_PALETTE_BLACK_HOT,
    SEEKCAMERA_COLOR_PALETTE_SPECTRA,
    SEEKCAMERA_COLOR_PALETTE_PRISM,
    SEEKCAMERA_COLOR_PALETTE_TYRIAN,
    SEEKCAMERA_COLOR_PALETTE_IRON,
    SEEKCAMERA_COLOR_PALETTE_AMBER,
    SEEKCAMERA_COLOR_PALETTE_HI,
    SEEKCAMERA_COLOR_PALETTE_GREEN,
    SEEKCAMERA_COLOR_PALETTE_USER_0,
    SEEKCAMERA_COLOR_PALETTE_USER_1,
    SEEKCAMERA_COLOR_PALETTE_USER_2,
    SEEKCAMERA_COLOR_PALETTE_USER_3,
    SEEKCAMERA_COLOR_PALETTE_USER_4,
} seekcamera_color_palette_t;
```

The `seekcamera_get_color_palette()` returns the active palette being used. The default palette is `SEEKCAMERA_COLOR_PALETTE_WHITE_HOT`. Additionally, the `seekcamera_color_palette_get_str()` API can be used to get the string name of a color palette.

Examples

Here are examples of each palette with a hot mug on the left and an ice cold mug on the right. Each picture was taken with the HistEq AGC mode and the default settings.

Figure 4. White Hot:- Traditional color palette and the SDK's default palette.

Figure 5. Black Hot: Reverse of the White Hot. Useful when there is sky in the scene.

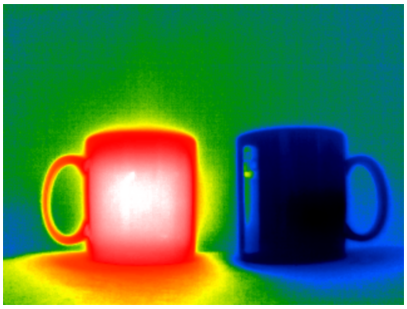


Figure 6. Spectra: Traditional palette used in Test and Measurement. Good at highlighted the source of the heat.

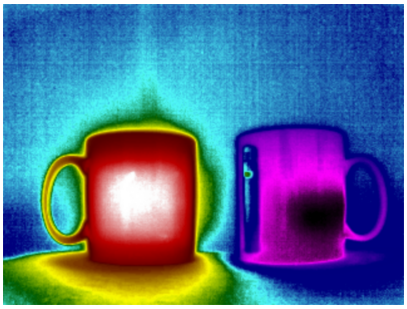


Figure 7. Prism: Similar to Spectra but with more layers.

Figure 8. Tyrian: Seek's modern palette version of iron.

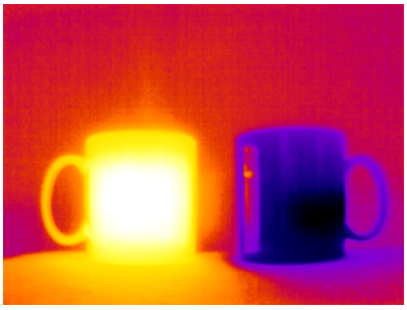


Figure 9. Iron: Legacy palette from ther early days of thermal imaging.

Figure 10. Amber: Newer palette that is generally accepted as pleasing to the eye.

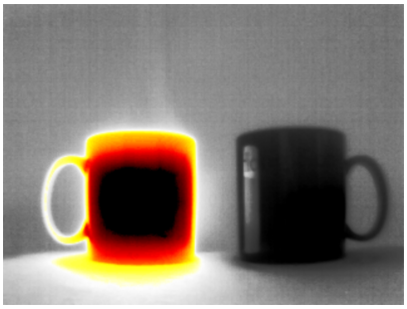


Figure 11. Hi: Very good at highlighting the hot spots.

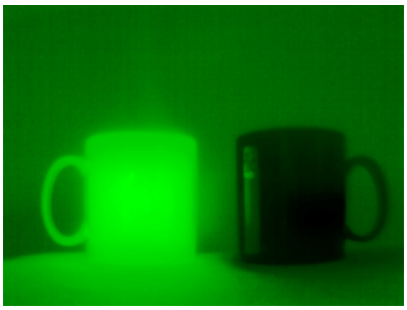


Figure 12. Green: Similar to Amber but with green.

User Palettes

Each color palette is basically a Look-Up Table (LUT) that has 256 entries. Each entry is 32-bits that is divided into 8-bits of alpha, 8-bits of red, 8-bits of green, and 8-bits of blue. Here is the actual data structure for an entry from `seekcamera.h`.

```
typedef struct seekcamera_color_palette_data_entry_t
{
    uint8_t b;
    uint8_t g;
    uint8_t r;
    uint8_t a;
} seekcamera_color_palette_data_entry_t;
```

The SDK allows the application to specify five "user" color palettes via the `seekcamera_set_color_palette_data()` API. Here is an example of setting the LUT data.

```
#define LUTCOLORS 256
#define ARGB(r, g, b) (0xff000000        |
                       (uint32_t)r << 16 |
                       (uint32_t)g << 8  |
                       (uint32_t)b)

seekcamera_color_palette_data_t white[LUTCOLORS] = {
    ARGB(0, 0, 0),
    ARGB(1, 1, 1),
    ARGB(2, 2, 2),
        .
        .
```

```
        .
    ARGB(253, 253, 253),
    ARGB(254, 254, 254),
    ARGB(255, 255, 255),
};
```

Then after a camera has connected, here is an example of creating the `SEEKCAMERA_COLOR_PALETTE_USER_0` with the above `white` LUT and set it to the active palette.

```
    status = seekcamera_set_color_palette_data(camera,
                SEEKCAMERA_COLOR_PALETTE_USER_0,
                (seekcamera_color_palette_data_t *)white);

    status = seekcamera_set_color_palette(camera,
                SEEKCAMERA_COLOR_PALETTE_USER_0);
```

AGC Modes

The image frames can be displayed in two different AGC modes: Linear and HistEq. The SDK's `seekcamera_set_agc_mode()` API can be used to set the AGC mode for a camera. The default is `SEEKCAMERA_AGC_MODE_HISTEQ`.

Both AGC modes have their merit and it really depends on your application to determine which one is better for you. Additionally, some environments and color palettes work better with Linear AGC mode and others work better with HistEq AGC mode. Let's look at each mode in more detail now.

Linear Mode

When the AGC mode is linear (`SEEKCAMERA_AGC_MODE_LINEAR`) and the default lock mode (`SEEKCAMERA_LINEAR_AGC_LOCK_MODE_AUTO`), the palette is spread linear across the entire scene. So the lowest temperature pixel(s) get the color from the zeroth entry in the LUT (e.g. ARGB(0, 0, 0) which is white[0] above). Similarly, the highest pixel(s) get the 255th entry (e.g. ARGB(255, 255, 255) which is white[255] above). Note: since the scene is dynamic, the min and max temperature are dynamically determined for each frame and the palette is spread accordingly.

A user can also hard fix the upper and lower limits of the linear color mapping by temperature. Set the upper and lower limits using the `SEEKCAMERA_SET_LINEAR_AGC_LOCK_MIN` & `SEEKCAMERA_SET_LINEAR_AGC_LOCK_MAX`.

In `SEEKCAMERA_LINEAR_AGC_LOCK_MODE_MANUAL` mode, the palette colors are spread across the locked minimum and maximum temperature values. If `SEEKCAMERA_LINEAR_AGC_LOCK_MODE_MANUAL_MIN` is used, the palette is spread across the locked minimum and the dynamically determined maximum. Similarly, if `SEEKCAMERA_LINEAR_AGC_LOCK_MODE_MANUAL_MAX` is used, the palette is spread across the locked maximum and the dynamically determined minimum.

Since the Linear AGC mode spreads the LUT evenly across the entire scene, this mode may not be ideal for thermally flat scenes or when there is a significant temperature range in the scene. Please

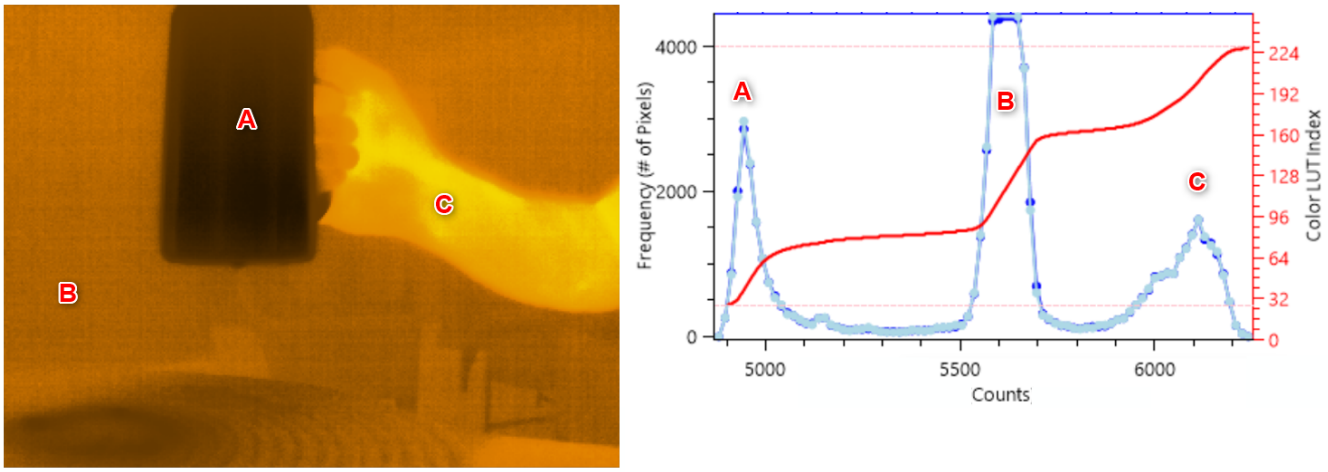
refer to Linear vs HistEq Examples for more details.

HistEq Mode

The `SEEKCAMERA_AGC_MODE_HISTEQ` is a bit trickier. Instead of spreading the palette linearly across the scene, the SDK determines by grouping the pixels (via histograms) the best way to spread the colors across the scene. This generally gives better granularity to the image since all the 256 LUT entries will be in the scene. The default settings for the HistEQ algorithm are good for most applications; however, the SDK has several Advanced Control APIs that allow a customer to fine tune the HistEQ imaging algorithm to their specific application and enviornment.

Advanced Control APIs

First let's start by looking at what the histogram looks like for a sample image. Below is an image with a human holding a cold mug in a indoor room. The x-axis is the sensor count for a pixel. A lower count corresponds to a lower temperature and a higher count corresponds higher temperature. The y-axis is number of pixels at the corresponding X-axis value.



The "A" peak on the histogram corresponds to the pixels that make up the icy mug. The "B" peak is the ambient temperature of the room. The wider "C" peak corresponds to human arm.

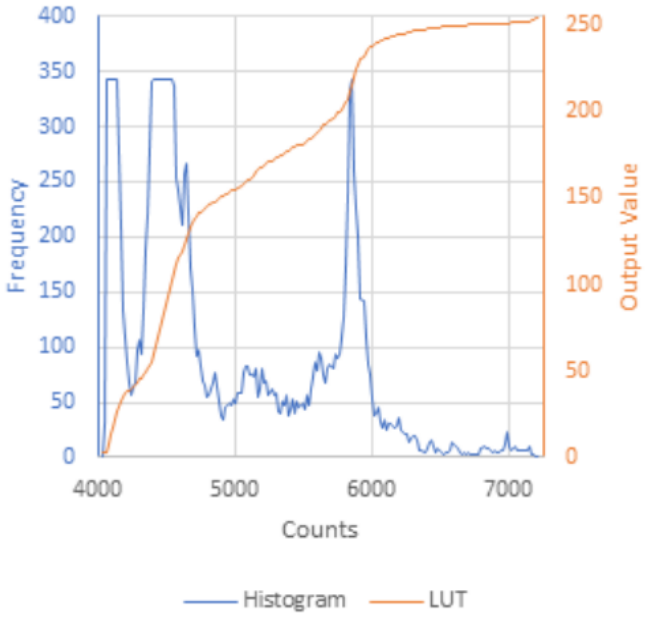
Notice that the entries of the palette (LUT) entries are spread more heavily in the "A", "B", and "C" peak regions. There are fewer palette entries assigned to the valley's between these peaks.

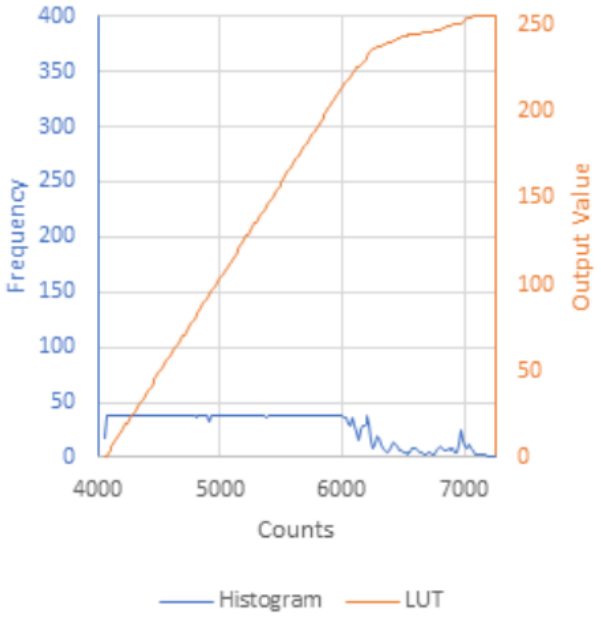
AGC Plateau

The AGC plateau API `seekcamera_set_histeq_agc_plateau()` limit the number of pixels that can be assigned to a single histogram bin. This is necessary when a large area of the scene, such as the sky, is very uniform and should be assigned a small number of colors. The default value of 0.05 (5%) means that a maximum of 5% of the total number of pixels can be assigned to a single bin. Bins are capped at this height and clipped pixels are not re-distributed in the histogram.

The image below on the left is a high-contrast scene with an optimally-tuned Plateau Value. The right figure is a plot with a very small Plateau Value. The LUT is very linear, which results in an image on the right that loses some detail within each object (flatter coffee cup, water bottle, and background).

Here are the histographs generated from these images. The histogram on the left has higher peaks and therefore more LUT entries associated with them. The histogram on the right (with the lower AGC plateau setting) does not have the peaks and therefore fewer LUT entries where the peaks used to be.





AGC Gain

When using Seek Thermal's AGC, the 'gain' portion essentially controls the maximum number of output values that may be assigned for each count of input data. Typically, gain is limited so that intense and concentrated data resulting in histogram peaks do not use up an extraordinary amount of output count values; reducing the contrast of in the rest of the image. For example, assigning too many colors to a low contrast scene would only increase noise. Conversely, a scene with a lot of content that is gain-limited may appear very washed out. Seek allows its users to finely adjust gain using a variety of settings explained in this section.

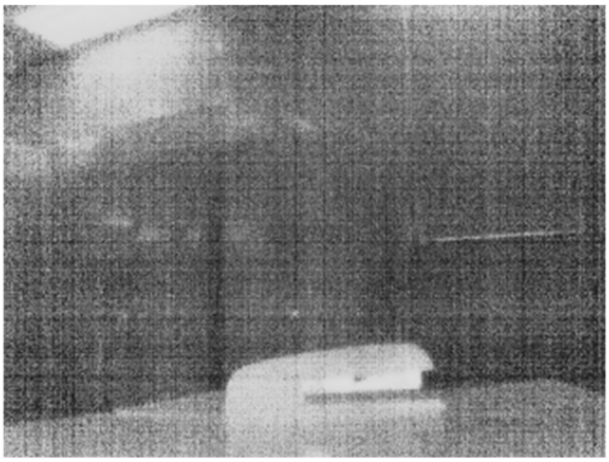
Gain, GainLimit and GainLimitFactor

Gain Control is broken into two pieces: GainLimit and GainLimitFactor. GainLimit is a single setting that controls the maximum gain (slope) of output values per count. GainLimitFactor adds additional flexibility in automatically reducing gain as a function of the histogram width. This additional control allows the user to tune the GainLimit value for a given application, and then allow the algorithm to reduce gain as the scene content decreases.

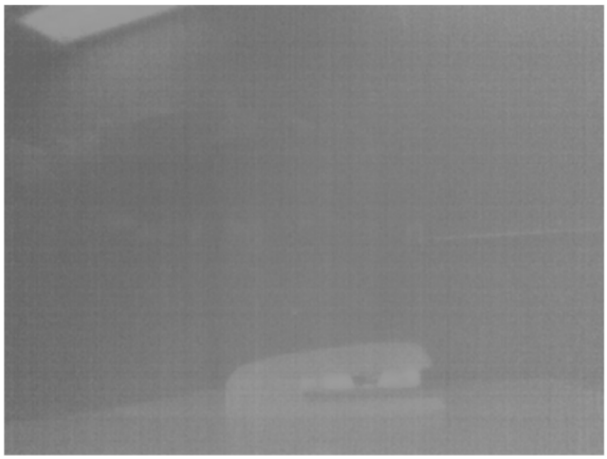
GainLimit

The `seekcamera_set_histeq_agc_gain_limit()` API limits the amount of output bits assigned to a single bin. The unit is in terms of the number of output values per sensor count. The default value of 0.45 means that no more than 0.45 of the available 256 output values may be assigned to a single sensor count. 4096 bins by default resulting in a bin width of 16 counts (216/4096), which is equivalent to 7.2 (0.45*16) output values that may be assigned to a single bin.

While this parameter may be difficult to understand, it plays a critical role in how output values (and color) may be distributed across a scene. The examples below show how GainLimit effects a “dull” or low-contrast scene. The optimal setting for this scene would likely be a GainLimit setting slightly higher than the right side.



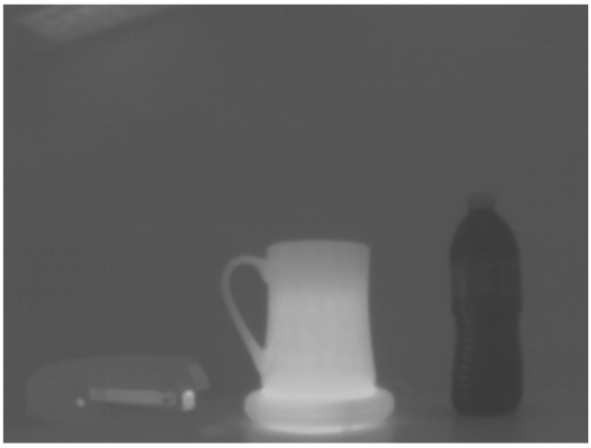
High Gain Limit in Low Contrast Scene



Low Gain Limit in Low Contrast Scene

The below images show the difference between low and high GainLimit settings in a high-contrast scene. Note that right side appears somewhat washed out, while in left side the background wall and light have more contrast.

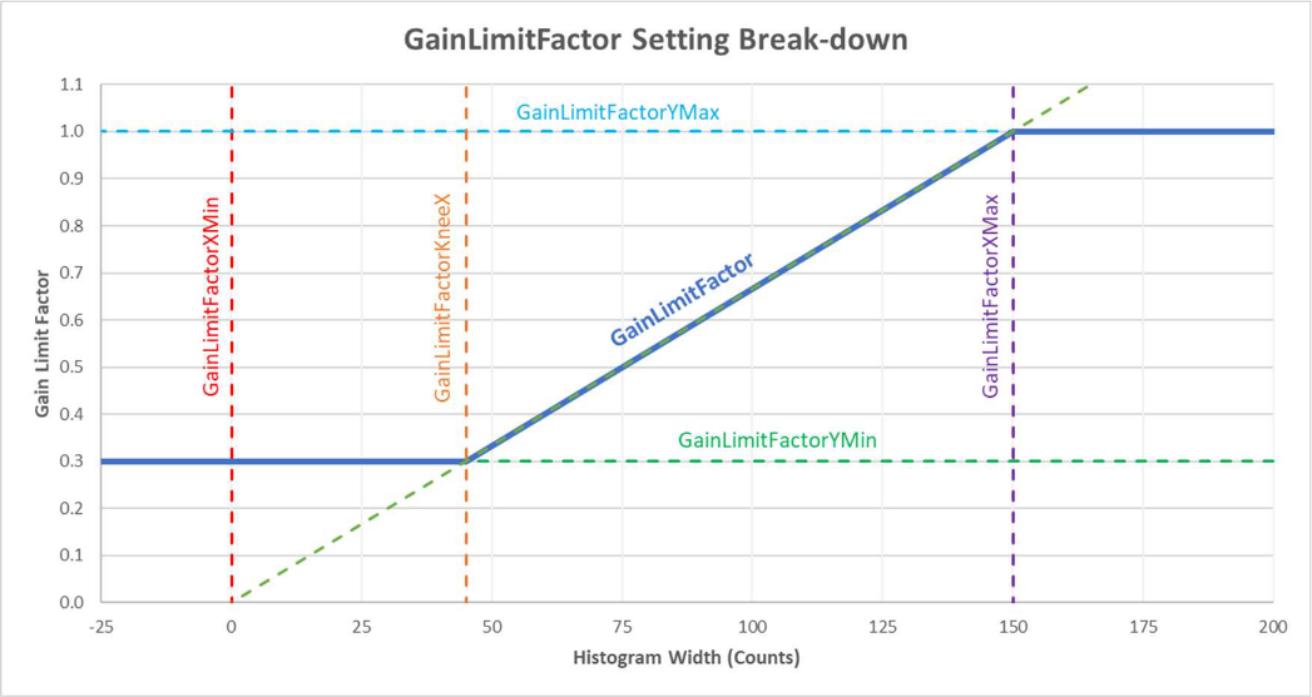
High Gain Limit in High Contrast Scene



Low Gain Limit in High Contrast Scene

Gain Limit Factor

The GainLimitFactor is a setting to alter the transfer function gain for particularly low-contrast scenes where noise may adversely affect the image. GainLimitFactor is controlled through a set of four control settings. In addition to those control settings, there are a handful of read-only settings that are useful when tuning GainLimit and GainLimitFactor.



GainLimitFactor Parameters

Typically, low-contrast scenes, such as a wall or empty sky, are relatively uniform and would be assigned too many output values resulting in a very noisy image. In this case, when a scene has very little content, it results in a narrow histogram. The GainLimitFactor collection of settings will smoothly reduce gain, thus reducing the number of output values used. This helps to make low-contrast scenes appear flat while maintaining the flexibility of having the GainLimit parameter set to a higher value to maintain higher contrast in non-flat scenes.

The `seekcamera_set_histeq_agc_gain_limit_factor_mode()` API is used to set the GainLimitFactor.

When GainLimitFactor is 1, the GainLimit value is multiplied by the GainLimitFactor to calculate the actual gain value used. When 0, the GainLimit value is unaltered and used as-is for HistEQ gain. The default is 1.

The `seekcamera_set_histeq_agc_gain_limit_factor_xmax()` sets the width of the histogram where the GainLimitFactor will begin to kick in. The default value of 150 counts means that for scenes where the histogram content is all contained within 150 counts or less, GainLimit will be scaled down. The amount of scaling is proportional to the width of the histogram such that a histogram width of GainLimitFactorXMax (150 by default) or higher will result in no scaling, and a histogram width of anything below GainLimitFactorXMax would result in a GainLimitFactor of GainLimitFactorYMin (0.3 by default).

The `seekcamera_set_histeq_agc_gain_limit_factor_xmax()` controls the minimum value for GainLimitFactor. Its range is 0.0 to 1 with a default value of 0.3 (which represents 30%).

The `histeq_agc_gain_limit_factor` is the `seekcamera_frame_header_t` structure gives the read-only GainLimitFactor. The GainLimitFactor is the value multiplied by GainLimit to calculate the maximum gain allowed when creating the histogram transfer function. This value can be used to determine if GainLimitFactor is enabled and affecting the gain.

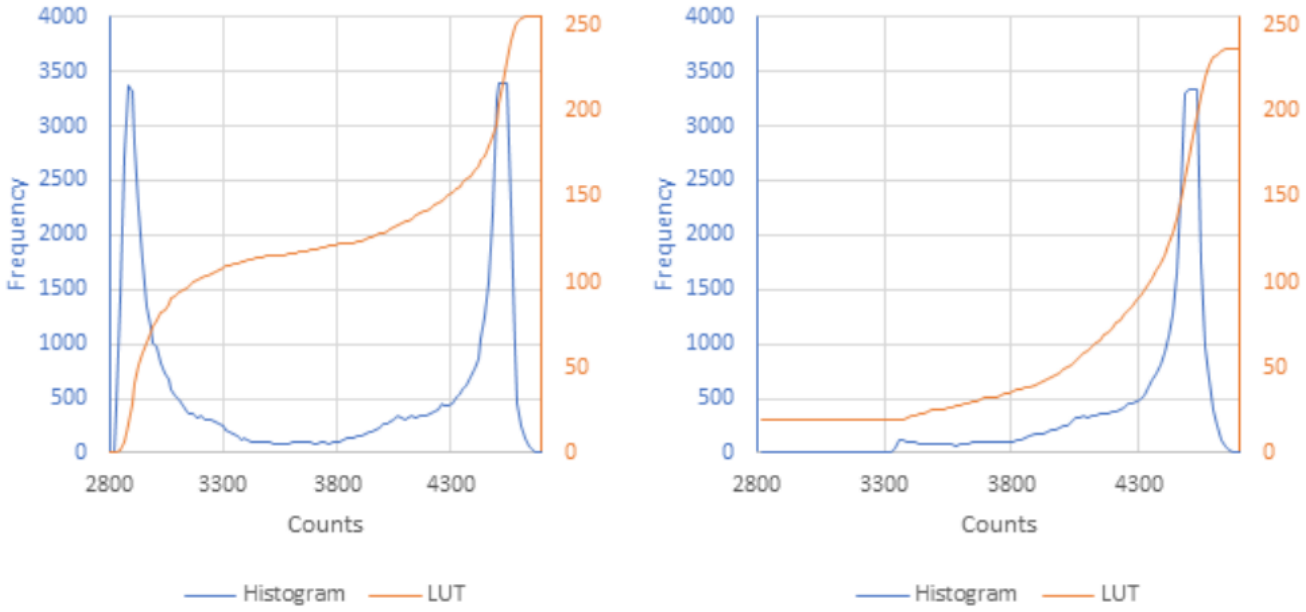
Trims

The `seekcamera_set_histeq_agc_trim_left()` API allows the application to removes outliers from the left side of the input scene's histogram. The default of 0.001 trims the left 0.1 % of the scene histogram and leaves the remaining 99.9 % of image content to feed HistEQ. Setting this parameter helps to prevent situations where a small number of outlier pixels are adversely affecting the scene's contrast.

The `seekcamera_set_histeq_agc_trim_right()` API allows the application to removes outliers from the right side of the input scene's histogram. The default of 0.001 trims the right 0.1 % of the scene histogram and leaves the remaining 99.9 % of image content to feed HistEQ. Setting this parameter helps to prevent situations where a small number of outlier pixels are adversely affecting the scene's contrast.

Setting the trims while in AGC HistEq Mode are similar to the min/max settings while in the AGC Linear Mode.

Here is an example with the left trim set to .40 (40%). Note how the LUT entries that where in the first 40% on the left are now redistributed to the other 60% of the scene on the right image.



Alpha Time

When scene content rapidly changes, the HistEQ output can flash and distract the user. The `seekcamera_get_histeq_agc_alpha_time_seconds()` API blends the current frame's histogram with the previous frame's histogram. The setting controls the length of one time-constant, where three time-constants typically result in the new scene being completely blended in. Therefore, the default setting of 1/3 will blend one second of previous frames, such that if a completely new scene is presented to HistEQ, it will take approximately 1 second for the histogram to fully incorporate the new scene. A value of zero effectively disables alpha blending.

HistEQ ROI

The histogram eualization region of interest commands allow the application to remove unwanted data from the histogram alogrithm calculations. This typically allows more scene content to be

seen, more contrast, and more detail to the overall image. The `seekcamera_set_histeq_agc_roi_enable()` command will enable the set region of interest. By default the historgram uses the entire frame to calulate its histogram equalization. The `seekcamera_set_histeq_agc_roi_top()` and `seekcamera_set_histeq_agc_roi_left()` commamnds will set the top left position of where the region of interest will start from. The `seekcamera_set_histeq_agc_roi_height()` will set where the bottom of the region stops. The `seekcamera_set_histeq_agc_roi_width()` will see the where the right side of the region stops at.

Changing the HistEQ ROI is especially useful in outdoor fixed mounted applications. In this application a camera may be pointed in a position where there is a lot unwanted data like sky or hot concrete. By shinking the region that the histogram alogrithm uses to only include the content of value in the scene we can an improvement in the overall imagery. In the example below, the image on the left uses the full frame for the HistEQ algorithm, while the iamge on the right only uses the bottom 50% of the frame for the HistEQ algorithm.

Full Frame ROI / HistEQ ROI Disabled

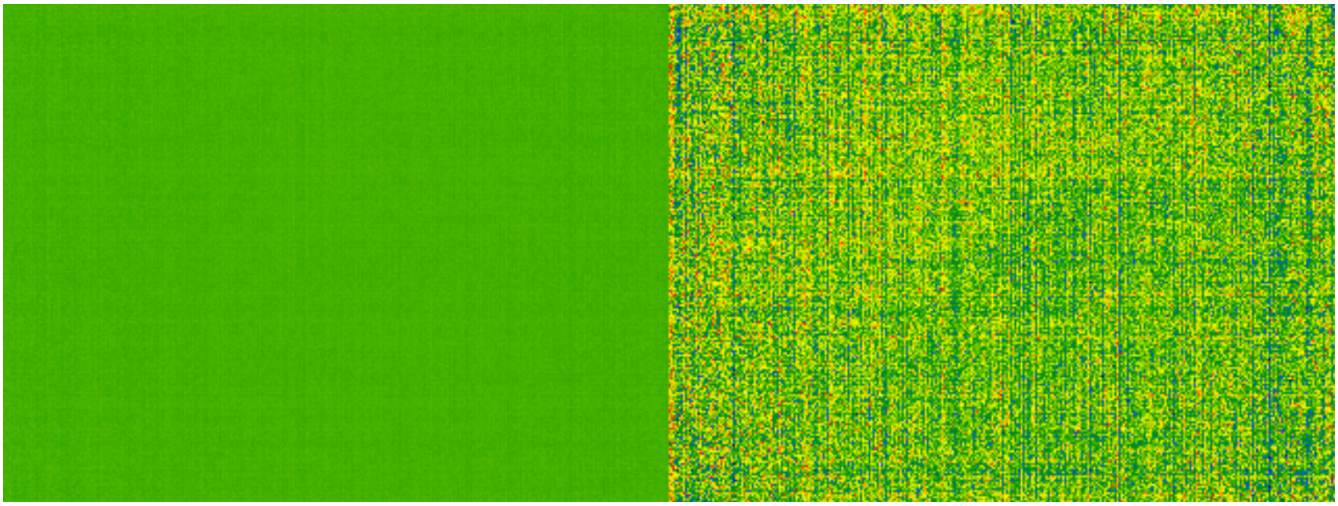
Bottom Half of Frame ROI

Linear vs HistEq Examples

Let's compare some Linear and HistEq images. In these images, we're using the default values for both Linear and HistEq modes. As noted in the above section, there are additional APIs to obtain more control of the images.

Thermally Flat Scene

Here is an example of a image of a wall with a small vertical seam on the right side of the scene that has a slightly different temperature. The left image is in HistEq AGC mode (with Spectra color palette) and the right side is in Linear AGC mode (with the same Spectra palette). Since the wall is basically the same temperature across the entire scene, the linear mode's image seems chaotic. This is because the SDK is trying to spread the entire color palette across a surface whose temperature only varies a few tenths of a degree.



Extreme Temperatures Scene

Here is a scene with four different temperatures

- A black body device with a circle of ~300C
- A hand of ~36C
- A wall of ambient temperature ~20C
- A mug with ice water ~1C

On the left side, the image was recorded with HistEq mode and the Black Hot palette. Notice you can pick out all the objects easily. The right side (Linear mode), you can faintly see the mug and hand. You can clearly see the hot circle of the 300C black body. Since the 256 color entries are spread across the entire scene, the cooler objects (mug, wall, and hand) get fewer color entries when in Linear mode. The HistEq mode sees the groupings and adjust accordingly. You can even see a ring on the hand and see that the bottom of the person's hand is getting warmed up from the black body device on the left image.

Here both modes have their merit. If you were trying to see a person in a fire, the HistEq mode would be better. If you were trying to find a hotspot and did not care about anything else, the Linear mode would be better.

Please note that the above images are possible with Seek Thermal Cameras since they have dual-

gain smart pixels. Dual-gain pixels provide the ability to resolve colder objects in a frame (room temperature or below) while resolving hotter items (fire up to 600C) in the image.

Pipeline Modes

The image frames can be displayed in three different pipeline modes: Lite, Legacy, and SeekVision. The SDK's `seekcamera_set_pipeline_mode()` API can be used to set the pipeline mode for a camera. The default is `SEEKCAMERA_IMAGE_LEGACY`.

All pipeline modes have their merit and it really depends on your application to determine which one is better for you. Additionally, some environments work better with SeekVision mode and others work better with Legacy or Lite mode. Let's look at each mode in more detail now.

Lite Mode

The Lite Pipeline mode has the minimum image processing correction filters and colorization dynamics.

Legacy Mode

The Legacy Pipeline mode includes image processing filters to address scene-based non-uniformities and motion. This is the default pipeline mode for most cores, without a special build request.

SeekVision Mode

The SeekVision Pipeline mode is the newest addition available to Mosaic cores. This includes image processing filters to reduce more background noise, while enhancing details and colorization contrast. This mode will dynamically adjust filter and AGC settings based on image content and the customer will not be able to update AGC settings in this Pipeline mode and will return the SEEKCAMERA_ERROR_NOT_SUPPORTED error.

Below are some example images of the SEEKCAMERA_IMAGE_LITE Pipeline on the left vs the SEEKCAMERA_IMAGE_SEEKVISION Pipeline on the right.

By default, the SeekVision Pipeline mode is not enabled since it may require additional processing power and is only available for Seek Thermal Mosaic OEM Cameras and will return the SEEKCAMERA_ERROR_NOT_IMPLEMENTED error with other cameras. Here is an example of setting the SeekVision Pipeline mode

```
seekcamera_set_pipeline_mode(camera, SEEKCAMERA_IMAGE_SEEKVISION);
```

Partially Processed Frames

Both the `SEEKCAMERA_FRAME_FORMAT_CORRECTED` and `SEEKCAMERA_FRAME_FORMAT_PRE_AGC` frame format are available in the SDK. In general both formats are intended for advanced users that are familar with thermal imaging and want to do custom imaging.

- `SEEKCAMERA_FRAME_FORMAT_CORRECTED`: The least modified count values from the sensor. This format is useful for users that want to perform their own noise filtering.
- `SEEKCAMERA_FRAME_FORMAT_PRE_AGC`: Useful for users that want to do their own colorization and AGC handling.

Shutter

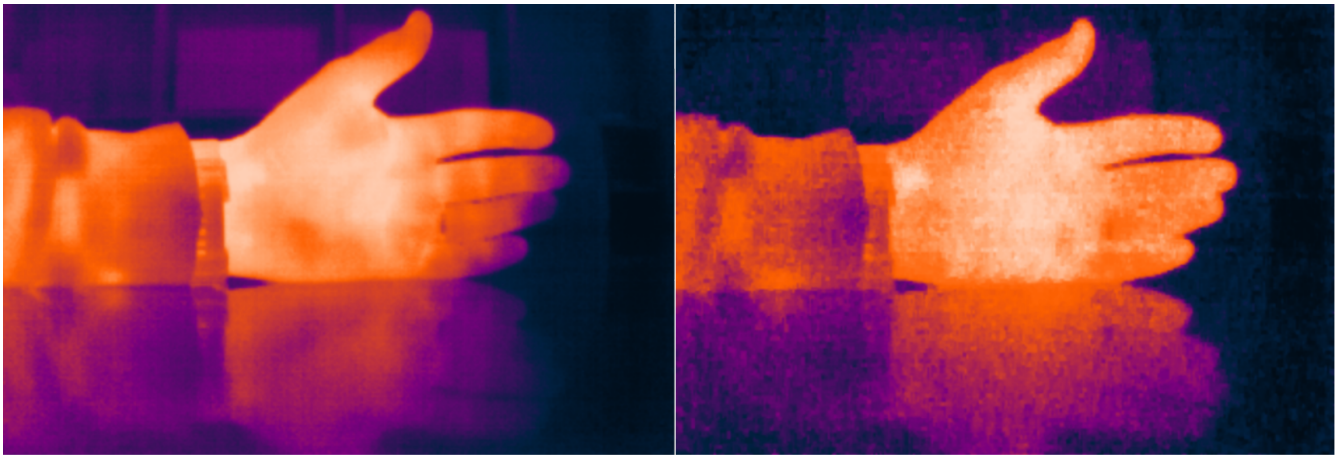
Most Seek Thermal camera products use an internal shutter: OEM Mosaic Core camera, Compacts, and Seek InpsectionCAM. The shutter helps the camera perform a Flat Field Correction (FFC). which corrects any potential pixel drift that might have occurred due to operating conditions. When the shutter is closed, the core performs the FFC.

The Micro Core cameras do not have a shutter. They are designed with a newer technology that does not require a shutter.

The shutter makes a small audible clicking sound when the shutter closes and opens. The shutter closure duration depends on the frame rate of the camera. In general, a shutter closure impacts 7 to 8 frames regardless of the frame rate. Note: the SDK filters out all the frames that are impacted by a shutter closure. Therefore a shutter event is more noticeable on the <9Hz cores when compared to the fast frame cores.

By default the SDK enables the automatic shutter for a camera.

Below the image on the right shows what can occur if the shutter is disabled for too long. The left side was taken during normal shutter operation. The right image is after disabling the shutter for an extended duration.



Calibration

All Seek Thermal cameras are calibrated at the factory. Each unique calibration data is loaded into the sensor and resides with that camera for its life. There should be no need to redo the factory calibration.

Secondary Calibration

Customers can do a 'secondary calibration' on top of the factory calibration. The simplest secondary calibration is to simply apply a global offset to the entire thermography frame via the `seekcamera_set_thermography_offset()` API in the SDK. The offset can even be stored on the camera itself and retrieved and set on power up. Please refer to Application Specific Storage below for more details.

Application Specific Storage

The SDK allows the application to store and retrieve data from the flash memory on the co-processor. This allows an application to store data that might be specific to the core. For example, if there is a secondary calibration performed on the camera, any resulting data can be stored on the co-processor associated with that camera.

Note: since the data is stored on the co-processor's flash instead of the actual sensor core, care must be taken to keep the sensor core together with the co-processor if there is sensor core specific information stored into the flash.

There are three seperate application specific regions on the core's co-processor. Each region is 64KB in size. The three regions are named `SEEKCAMERA_APP_RESOURCES_0`, `SEEKCAMERA_APP_RESOURCES_1`, and `SEEKCAMERA_APP_RESOURCES_2`. The `seekcamera_store_app_resources()` API allows an application to store application specific data into these regions. The format of the data is entirely up to the application and will not be modified by the SDK.

The `seekcamera_load_app_resources()` API loads the contents of the regions back into an application provided buffer.

The `seekcamera_store_app_resources()` and `seekcamera_load_app_resources()` APIs can only be called after the camera is connected but before the `seekcamera_capture_session_start()` or after the `seekcamera_capture_session_stop()` APIs. They are not allowed during an active session because the writing/reading to the co-processor's flash could impact the throughput of the sensor frames.

Both `seekcamera_store_app_resources()` and `seekcamera_load_app_resources()` APIs support the ability to have the application specify a progress callback. As the store or load progresses, this callback will be called with the percent progress (0 to 100) and the `user_data` value supplied. The callback is optional (using `NULL` is allowed) but might be helpful since the load/store API calls will be blocked until the action is completed.

Filters

The SDK also comes with filters that impact all frame formats.

Flat Scene Correction

The Flat Scene Correction (FSC) is a filter that is intended to help correct any artifacts introduced when placing an OEM core into a vendors final assembly. It is intended to only be done once after the customers final assembly. Please refer to Flat Scene Correction Application Note page on the Seek Thermal's Developers Portal for a more complete discussion of this feature.

Gradient Correction

The gradient filter is a dynamic filter that corrects gradiention. When the SDK detects thermally flat objects and the gradient filter is enabled, the SDK attempts to correct any incorrect artifacts in the image. This works very well with most handheld applications and products.

By default, the gradient filter is disabled. Here is an example of enabling the filter

```
seekcamera_set_filter_state(camera, SEEKCAMERA_FILTER_GRADIENT_CORRECTION,
                            SEEKCAMERA_FILTER_STATE_ENABLED);
```

Sharpen Correction

Seek offers customers the ability to use Seek's legacy sharpening filter while using the LITE or LEGACY pipeline modes. The sharpen filter is intended to highlight the details in the image. This filter is a specialized high-pass filter to accentuate details while attentuating lower frequency noise.

By default, the sharpen filter is disabled. Here is an example of enabling the filter

```
seekcamera_set_filter_state(camera, SEEKCAMERA_FILTER_SHARPEN_CORRECTION,
                            SEEKCAMERA_FILTER_STATE_ENABLED);
```

Core Specific

The Seek Thermal SDK v4.x works on the OEM cores (Mosaic and Micro Cores) and Seek Compacts. The following is a table that denotes any core specific APIs or functionality

Table 1. Core Specific Functionality

API/Functionality	**Core**
Shutter Management	Mosaic Cores, Compacts, InspectionCAM
Pairing: `seekcamera_store_calibration_data()`	Micro Core only