



SenXor Android SDK

Doc Version:	3.8
SDK Version:	v3.4.0
Date:	23 Apr 2025
Authors:	Mason Yan, Johnson Leung

Revision History

Revision	Date	Comment
1.0	4 Feb 2020	<ul style="list-style-type: none"> Initial release
2.0	30 July 2020	<ul style="list-style-type: none"> Add Contouring module
2.1	13 Aug 2020	<ul style="list-style-type: none"> Add Software License
2.2	15 Sep 2020	<ul style="list-style-type: none"> Add Fever Detector, helloworld sample code
3.0	18 Dec 2020	<ul style="list-style-type: none"> Add SenXorView, Example1 & 2, remove tensorflow lite dependency from app module
3.1	16 Aug 2021	<ul style="list-style-type: none"> Update SDK setup method, project structure, SDK APIs, remove Software License
3.2	23 Sep 2021	<ul style="list-style-type: none"> Sync up doc with SDK v3.1.0
3.3	22 Dec 2021	<ul style="list-style-type: none"> SDK v3.2.0: Add dual camera overlapping in Example3; CBT support multiple ROI
3.4	10 Mar 2022	<ul style="list-style-type: none"> SDK 3.2.1: Fix various issues Fix memory leak issue on Senxor Library Fix Multiple USB, runtime plug and unplug Improve CBT accuracy on human modelling
3.5	30 Nov 2022	<ul style="list-style-type: none"> SDK 3.2.3: Add Cougar Rotate support Update CNN denoise model with less strength (more details preserved) Fix SenxorView issue on setup and start
3.6	8 Dec 2022	<ul style="list-style-type: none"> Minor release to add the old Xpro VID (0416) and PID (B002) to the library and examples
3.7	29 Jun 2024	<ul style="list-style-type: none"> Update SenXorLib Android with the version same as SenXorProViewerV2 (SDK V3.3.0) Update the layout to be compatible with Android 14
3.8	05 Aug 2024	<ul style="list-style-type: none"> Update Software STARK implementation in SDK (only AAR binary changed)
3.9	23 Apr 2025	<ul style="list-style-type: none"> Included new methods Updated style in APIs session

Table of Contents

Revision History	2
Introduction	4
Project Setup	5
Importing library (Optional)	6
Wireless Debugging	8
Android 10	9
Android 11 or higher	10
Project Structure	11
USB serial interface	13
SDK APIs	14
SenXorView	15
SenXorEvkDevice	17
ByteParser	18
SerialInputParser	19
SerialInputProcessor	20
SenXorEvkCommand	21
SenXorEvkMessage	22
Bobcat	23
SenXorFrame	25
TemperatureUnit	26
Processor	27
Renderer	28
SenXorFrameCBT (Core Body Temperature)	30
Appendix	33
<i>Mapping Temperature to a Colourmap</i>	<i>33</i>
List of Color Palettes	34
<i>Contouring in Thermal Image</i>	<i>35</i>
Output format	35

Introduction

SenXor Android SDK is a software development kit contains examples and SenXor library for developers who want to build Android applications with SenXor devices. This SDK contains two parts: the library and sample code. Because the library handles all the low level operations, with a few line of code from library, the development time for developer when using SenXor is thus reduced.

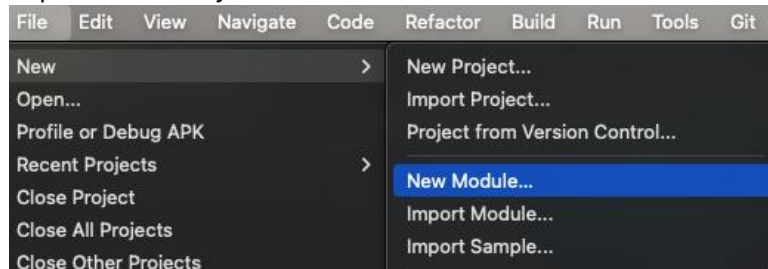
Project Setup

1. Launch Android Studio and click File -> New -> Import Project for SenXorAndroidSDK.
2. Build and run the application module. Please do not use Available Virtual Devices (AVD) but real device because external USB devices will not be detected in AVD.
3. Default project settings
`compileSdkVersion = 35`
`minSdkVersion = 29`
`targetSdkVersion = 34`
`Gradle version = 8.8`
`Android Plugin version = 8.5.0`
4. To install the application, connect to Android device and then click "Run".
5. Some devices may need to enable OTG when connecting the SenXor to the application.

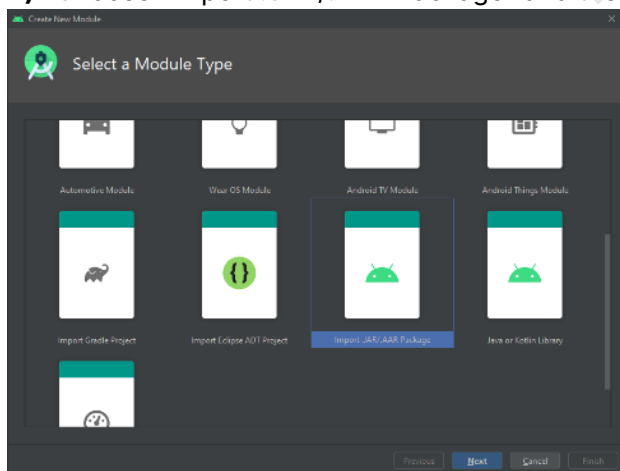
Importing library (Optional)

SenXor Android SDK is designed for the developers who wish to test the core functions of libsenxor Android. For developers who wish to develop their applications, they can utilise the libsenxor Android by importing libsenxor-android-release.aar file from SenXor Android SDK.

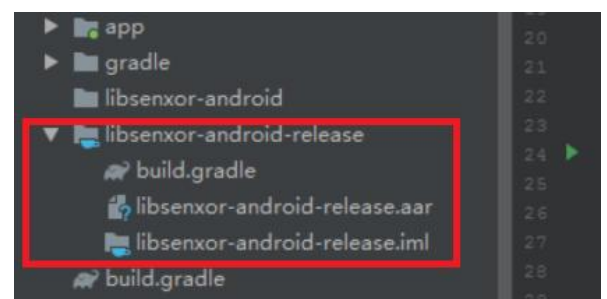
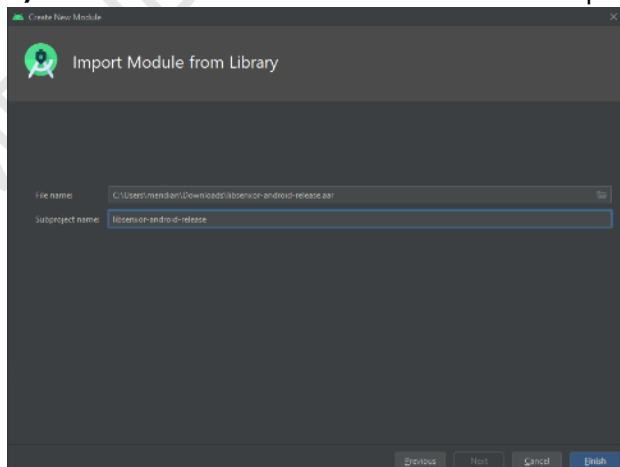
- 1) Obtain libsenxor-android-release.aar
- 2) Open your project in Android Studio
- 3) Import module by File → New → New Module...



- 4) Choose "Import .JAR/.ARR Package" and then press Next



- 5) Select libsenxor-android-release.aar to import library module



- 6) Add dependency in app gradle file:

```
implementation project (path: ':libsensor-android')
```

- 7) Add library module in settings.gradle:

```
include ':libsensor-android'
```

Wireless Debugging

Most of the Android devices contain only one USB port, to enable debugging when EVK is connected, wireless debugging is recommended.

To enable this feature, please check the following requirements are satisfied:

1. Android device and computer must be connected to the same network.
2. "Developer Options" is enabled in [Settings](#).
3. Android device and computer must be paired.
4. Android 11 or above (Officially supported) / Android 10 (Requires adb command)
5. Latest version of Android Studio

Android 10

Wireless debugging is not supported officially in Android 10 (SDK 29). To enable this feature, developers are required to establish connection manually via adb.

1. Connect device by USB.
2. Open console and navigate to adb directory
3. Enter command to verify whether adb can discover the device:

```
adb devices
```

4. If the device is discoverable, this message will be displayed.

```
List of devices attached
* daemon not running; starting now at tcp:5037
* daemon started successfully
da9db5e9 device
```

5. Start adb in TCP/IP mode. In this example, 5555 refers to the port and can be changed to any port.

```
adb tcpip 5555
```

6. Establish connection to the device:

```
adb connect <IP_address_of_device>:5555
```

For example, if the IP address of the device is 192.168.1.100. Then enter:

```
adb connect 192.168.1.100:5555
```

7. Verify whether adb is connected with the device:

```
adb devices
```

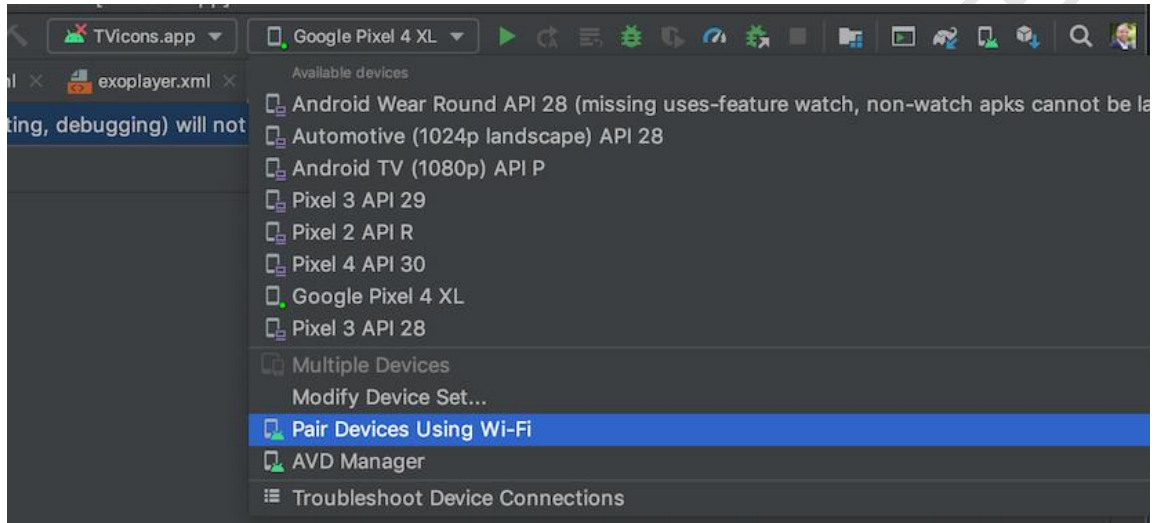
If the connection is established, adb will display the connected devices:

```
List of devices attached
192.168.1.100:5555 device
```

Android 11 or higher

Android 11 or above supports wireless debugging. Developers can enable this feature directly from "Developer Options" in "System settings"

1. On Android device, enable "Debugging over Wi-Fi".
2. In Android Studio, choose "Pair Devices Using Wi-Fi" from the run configurations dropdown menu

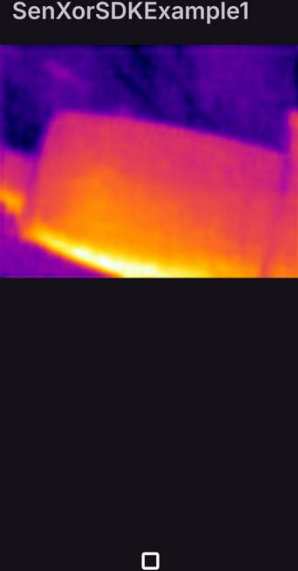
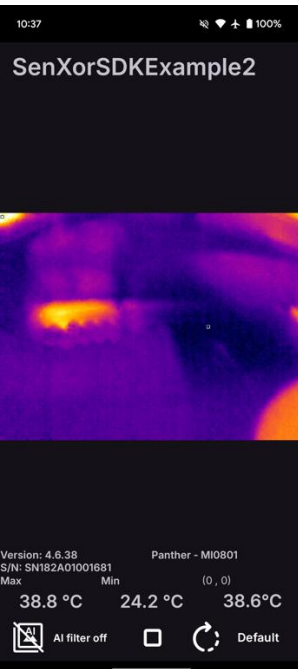


3. Pair the device by following the on screen instructions

Project Structure

SenXorAndroidSDK shipped with 3 application modules and a library module:

Application modules

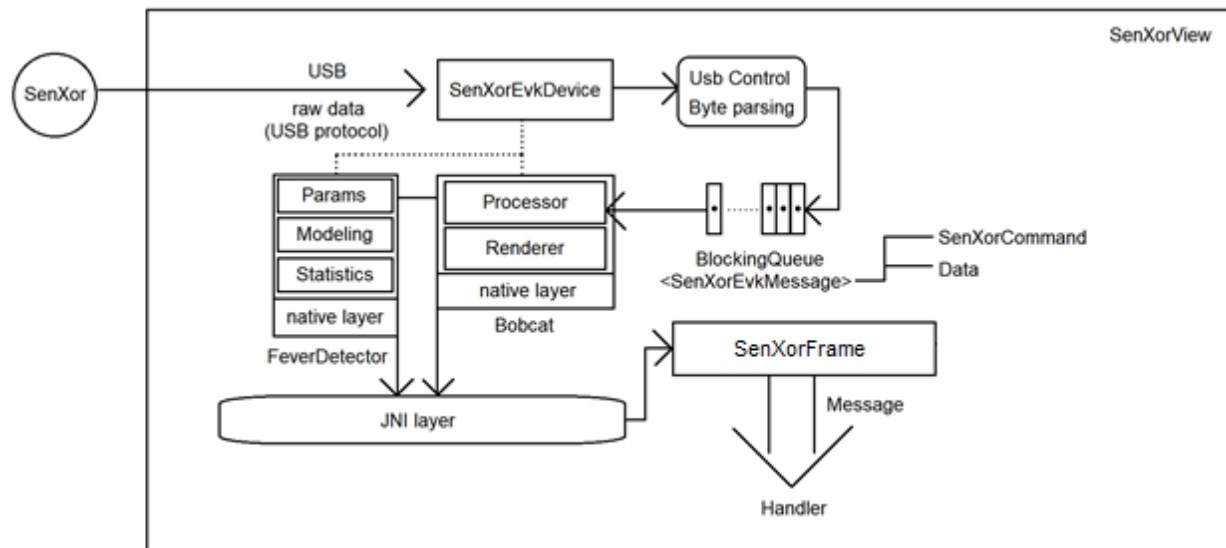
Example	Features demonstrated in the example
<p>1</p> 	<ul style="list-style-type: none"> Minimal coding by utilising libSenXor's wrapper class - SenXorView A button to start or stop capturing USB event handling using BroadcastReceiver
<p>2</p> 	<ul style="list-style-type: none"> Image enhancements: Deep learning filter and rotation Reading of product information: <ol style="list-style-type: none"> SenXor serial number Module type EVK's Firmware version EVK model Temperature reading: <ol style="list-style-type: none"> By specifying a coordinates (0,0 by default) Maximum temperature Minimum temperature
<p>Example 3</p>	<ul style="list-style-type: none"> Overlap thermal image and built-in camera by perspective transformation.
<p>Example 4</p>	<ul style="list-style-type: none"> Read a build-in firmware and update EVK firmware via USB-DFU protocol.

Library module

The core part of the SDK is the library module, `libsensor-android`. This foundations of the library is build on C++ with JNI and Java methods on the tops to realise the features:

- SenXor device interfacing
- Data streaming and handling
- Image processing
- Minimise coding by providing a wrapper class

The diagram below shows the processing flow of the library. SenXor devices (SenXor) are connected to Android device via USB. Communications between Android and SenXor are handled by `SenXorEvkDevice` which converts incoming byte stream into `SenXorEvkMessage`. Class `Bobcat` is a subclass of `SenXorEvkDevice` featuring image processing (Done by `Processor` class) and rendering (Done by `Renderer` class) in native layer (C++). Class `FeverDetector` is an optional module combining the result of `Processor` and `Renderer` to detect whether a person has fever. Finally, the processed data will be feed to the handler in the form of `Message`.



USB serial interface

SenXor device is using USB Virtual COM port (VCP) interface on Android platform with Type-C header. Vendor and product ID are provided for identifying SenXor USB device in `res/xml/device_filter.xml`.

Vendor ID: 0x0416

Product ID: 0xB002 (Bobcat),
0xB020 (Bobcat with TOF),
0x9393 (XCAM)

Baud rate: 115200

Data: 8 bit

Stop: 1 bit

Parity: None

Flow Control: OFF

SDK APIs

libsensor-android provides various APIs for developers to interface with SenXor. To enable high performance image processing, the majority of the APIs are written in C++.

SenXorView

SenXorView is the wrapper class of ImageView that controlling SenXorDevice within the class.

Usage

Add com.meridianinno.libsenxor_android.SenXorView in xml layout:

```
<com.meridianinno.libsenxor_android.SenXorView
    android:id="@+id/imageView_Thermal"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    app:layout_constraintDimensionRatio="H,80:62"/>
```

Parameters

SenXorView can be configured by adding app:parameter="option" in the xml layout:

Parameter	Available options
isAutoScaling	Boolean true - Enable auto scaling false - Disable auto scaling
isFeverDetect	Boolean true - Enable fever detection false - Disable fever detection
isNNEnable	Boolean true - Enable deep learning filter false - Disable deep learning filter
isPixelFilterEnable	Boolean true - Enable filtering false - Disable filtering
isRollingAveragingEnable	Boolean true - Enable rolling average false - Disable rolling average
isTemporalFilterEnable	Boolean true - Enable temporal filter false - Disable temporal filter
emissivity	Unsigned integer Range 0 - 100
feverdetector_blursize	Unsigned integer
feverdetector_minArea	Unsigned integer
feverdetector_threshold	Unsigned integer 0 - 255
fps	Unsigned integer
palette	Unsigned integer Range 0 - 15
pixelFilterSize	Unsigned integer
renderMaxTemp	Double
renderMinTemp	Double
rollingAveragingSize	Unsigned integer
rotate	Unsigned integer Range 0 - 1
temporalFilterSize	Unsigned integer

unit	Unsigned integer Range 0 - 2
------	------------------------------------

MERIDIAN INNOVATION LIMITED

SenXorEvkDevice

An abstract class for controlling a SenXor device attached to Android and handling the byte stream. Any SenXor should have its own class extending with SenXorEvkDevice.

Constructor

```
public SenXorEvkDevice(@NotNull Context context)
```

Constructor. Requires Context object as parameter.

Methods

```
public boolean isEmpty()
```

Check whether the SenXor is attached. Return true if it is not, false otherwise.

```
public boolean isOpen()
```

Check whether the SenXor is attached and connected. Return true if it is, false otherwise.

```
public void close()
```

Disconnect SenXor from device.

```
public void open()
```

Open a connection to SenXor that is attached to the device.

```
public void start()
```

Start image capturing.

```
public void stop()
```

Stop image capturing.

```
public void readCmd(String addr)
```

Read SenXor's register with address addr. The result is phrase and delivered in handler.

```
public void writeCmd(byte[] bytes)
```

Write a bytes of data, bytes, to SenXor.

```
public void writeCmd(String addr, int value)
```

Write to register with specified address addr and value value.

```
public void setFPS(int newFPS)
```

Set SenXor's frame rate (FPS). Parameter newFPS must be a positive integer.

```
public void setFPSdivider(int newDivider)
```

Set SenXor's frame divider. Parameter newDivider must be a positive integer, if newDivider is 0, the SenXor will capture thermal image in the highest frame rate.

ByteParser

A helper class of SenXorEvkDevice. This class provides methods to extract the command and data from the SenXor raw byte stream. The methods with reserved word native are JNI methods for accessing performance optimised C++ functions in the foundation of libsenxor.

Constructor

public ByteParser()

Methods

public boolean isDisposed()

Check whether the object is created. Return true if it is not, false otherwise.

public native void addBytesAndParse(byte[] inBytes)

Feed and parse command from a data stream inBytes from SenXor.

public native void disposeCppObject()

Free resource from memory. This method equivalent to delete in C++.

public native void initCppObject()

Create a C++ object. This method equivalent to new in C++.

public int getNumParsedMessagesAvailable()

Return the number of parsed messages available.

public native SenXorEvkMessage takeNextMessage()

Return the next parsed frame SenXorEvkMessage

SerialInputParser

A helper class of SenXorEvkDevice. It is also a Runnable threaded class that read the raw byte stream from queue queueIn and parse it in ByteParser. The result will return to queueOut.

Since this is a Runnable class, this must be executed by a thread.

Constructor

```
public SerialInputParser(BlockingQueue<byte[]> queueIn,  
BlockingQueue<SenXorEvkMessage> queueOut)()
```

Parameters queueIn and queueOut are BlockingQueue for raw byte stream and the result respectively. This also create a ByteParser object.

Methods

```
public SenXorEvkMessage getNewFrame(byte[] input)
```

Return a phased SenXorEvkMessage and get a new frame from raw byte stream.

```
public void run()
```

Implementing interface Runnable is used to create a thread for this object.

```
public void stop()
```

Stop the current task running in this object. Noted that this will not remove the runnable task from a thread.

SerialInputProcessor

A helper class of SenXorEvkDevice. It is also a Runnable threaded class that read the raw byte stream from queue queueIn and parse it in SerialInputParser. The result will deliver to main thread via msgHandler. Since this is a Runnable class, this must be executed by a thread.

Constructor

```
public SerialInputProcessor(BlockingQueue<byte[]> byteIn,  
BlockingQueue<SenXorEvkMessage> msgIn, Handler msgHandler)()
```

Parameters byteIn and msgIn are BlockingQueue for raw byte stream and the result respectively. msgHandler is a Handler for delivering the result to main thread.

Methods

```
public void setMsgHandler(Handler input)
```

Set new Handler input.

```
public void start()
```

Start and create a SerialInputProcessor thread.

```
public void stopAndReset()
```

Stop the current task running in this object. This will empty the queue byteIn and msgIn.

SerialInputParser will also stop by executing this method. Noted that this will not remove the runnable task from a thread.

SenXorEvkCommand

SenXorEvkCommand handles the USB serial protocol of SenXor. It contains index and cmdName and some common command sets such as start and stop continuous capturing.

The proper command set can directly control the SenXor by SenXorEvkDevice

writeCmd(bytes[]) or writeCmd(addr, value). For more details please refer to USB protocol documents.

Constructor

public SenXorEvkCommand(int index, String cmdName)

Parameters byteIn and msgIn are BlockingQueue for raw byte stream and the result respectively. msgHandler is a Handler for delivering the result to main thread.

Methods

public static SenXorEvkCommand fromString(String s)

Returns the EvkCommand object that matches the string s, or null if not found.

public static SenXorEvkCommand fromIndex(int idx)

Returns the EvkCommand object that matches the index idx, or null if not found.

public int getIndex()

Return the index.

public String getCmdName()

Return the command name.

SenXorEvkMessage

SenXorEvkMessage is the structured USB serial command set of raw byte data coming from SenXor. It is designed for easily parsing command set and data.

Constructor

public SenXorEvkMessage(SenXorEvkCommand command, byte[] data)

Create a SenXorEvkMessage object by specifying SenXorEvkCommand command and array of byte data .

public SenXorEvkMessage(SenXorEvkCommand command, String strData)

Create a SenXorEvkMessage object by specifying SenXorEvkCommand command and string strdata .

Methods

public SenXorEvkCommand getCommand()

Returns the SenXorEvkCommand.

public byte[] getData()

Returns the array of byte in a SenXorEvkMessage object.

public byte getDataAtIdx(int idx)

Returns the a byte in an array of byte in a SenXorEvkMessage object by specifying index idx.

public int getDataLength()

Return the length of the array of byte in a SenXorEvkMessage object.

Bobcat

This class is a subclass of SenXorDevice.

A class for controlling SenXorDevice and the SenXor thermal frame, it includes 4 elements: ByteParser, Processor, Renderer and SenXorFrameCBT. The pipeline of the SDK is to generate SenXorEvkMessage from ByteParser, process and render the data, optionally pass into SenXorFrameCBT, return the result frame to handler as message.

Constructor

public Bobcat(Handler uiHandler, Context context)

Create a Bobcat object. Parameter uiHandler enables processed result to be delivered to main thread and context specify the current application environment.

Methods

public Processor getProcessControl()

Returns the Processor object.

public Renderer getRendererControl()

Returns the Renderer object.

public SenXorFrameCBT getSenXorFrameCBT()

Returns the SenXorFrameCBT object.

public TemperatureUnit getUnitType()

Returns the TemperatureUnit object.

public boolean getIsRunning()

Returns the Processor object.

public static native void setFrameSize(int pCol, int pRow)

Set frame size.

public static native int getScale()

Returns the scale.

public static native void cppSetRendererParam(int paletteOrdinal, boolean isAutoScale, double scaleMin, double scaleMax)

Set parameter for the Renderer object. Parameter paletteOrdinal specify the colourmap style, isAutoScale turn autoscale on or off, scaleMin and scaleMax set the limits of the colourmap.

public static native void setRotate(int rotate)

Set rotation.

public static native void setScale(int scale)

Set scale.

public void close()

Disconnect SenXor from device.

public void open()

Open a connection to SenXor that is attached to the device.

public void setFrameSizeByType(int pModType)

Set the frame size according to module type pModType.

public void setUnitType(TemperatureUnit input)

Set temperature unit.

public void setTemperatureOffset(float input)

Set temperature offset by input.

public void start()

Start image capturing.

public void stop()

Stop image capturing.

SenXorFrame

This class that represents a processed colour mapped thermal frame and its properties. When a frame is captured from SenXor, the library processes and render the thermal frame. In the end of the process pipeline, the SenXorFrame object will be created containing the bitmap of the processed thermal frame.

Constructor

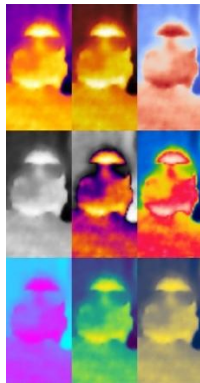
```
public SenXorFrame(int imageCols, int imageRows, int rotate, char[] header,  
char[] data, final byte[] imageData)
```

Create a SenXorFrame object. This constructor takes imageCols and imageRows as the dimension for the output image. Rotate specifies the rotation. The frame header and data is taken in parameter header and data. The last parameter, imageData contains RGB data of the image.

Methods

```
public ArrayList<Bitmap> getImageArr()
```

Returns the a ArrayList contains a list of colour bitmaps. The total bitmaps contain in the list is 9:



```
public Bitmap getImage()
```

Returns the a colour bitmap image

```
public char[] getData()
```

Returns the data in an image.

```
public char[] getHeader()
```

Returns the header in an image.

```
public double getDataAt(int x, int y)
```

Returns the value in an image by specifying the coordinate x and y.

```
public int getImageCols()
```

Return the number of column in an image.

```
public int getImageRows()
```

Return the number of row in an image.

TemperatureUnit

The library supports 3 types of units: KELVIN, DEGREE_C and DEGREE_F. They are used as ordinal start from index 0.

Constructor

This static class does not require a constructor.

Methods

public static String getUnitSymbol(int unitOrdinal)

Returns the temperature unit symbol. The available options for the parameter unitOrdinal is 0 - Kelvin, 1 - degree celsius, 2 - fahrenheit.

public static double convert(double input, TemperatureUnit unitInput, TemperatureUnit unitOutput)

Convert a value from one temperature unit to another. This method will take the original value input with its unit unitInput to target unit unitOutput.

Processor

This class represent the image processing component in libSenXor.

Constructor

public Processor()

Create a Processor object.

Methods

public boolean isDisposed()

Check whether the object is created. Return true if it is not, false otherwise.

public native void setIsDnnEn(boolean isDnnEn)

Enable or disable neural network based filter. Set isDnnEn to true to enable, false otherwise.

public native void setIsStarkEn(boolean isStarkEn)

Enable of disable Meridian STARK filter. Set isDnnEn to true to enable, false otherwise.

public native void setStarkFPSdiv(int starkFPSdiv)

Set STARK FPS. starkFPSdiv is the SenXor's frame divider

public native int getFilteredMax()

Returns the maximum temperature in a frame after filtering.

public native int getFilteredMin()

Returns the minimum temperature in a frame after filtering.

Renderer

This class is used to colorise the thermal data with preset color palettes. There are scaleMin and scaleMax which are the upper and lower boundary to map the color to thermal data in certain range. Auto scale will set the range referring to the min-max temperature of the frame. The preset palettes are listed in Appendix.

Constructor

public Renderer()

Create a Renderer object.

Methods

public native double getColorPaletterScaleMin()

Check whether the object is created. Return true if it is not, false otherwise.

public native double getColorPaletterScaleMax()

Enable or disable neutral network based filter. Set isDnnEn to true to enable, false otherwise.

public native void setColorPaletteNum(int newPaletteNum)

Set the colour map by specifying the colourmap index newPaletterNum.

public native void setColorPaletteIsAutoScale(boolean isAutoscale)

Enable or disable autoscaling in colourmapping. Set isAutoscale to true to enable, false otherwise.

public native void setColorPaletteScaleMax(short newPalatteScaleMax)

Set colour mapping upper boundary temperature.

public native void setColorPaletteScaleMin(short newPalatteScaleMin)

Set colour mapping lower boundary temperature.

public native void setFlip(int flip)

Flip image. Set parameter flip to 1 - the x-axis, 2 - the y-axis, 3 - x and y axes, 0 - Disabled.

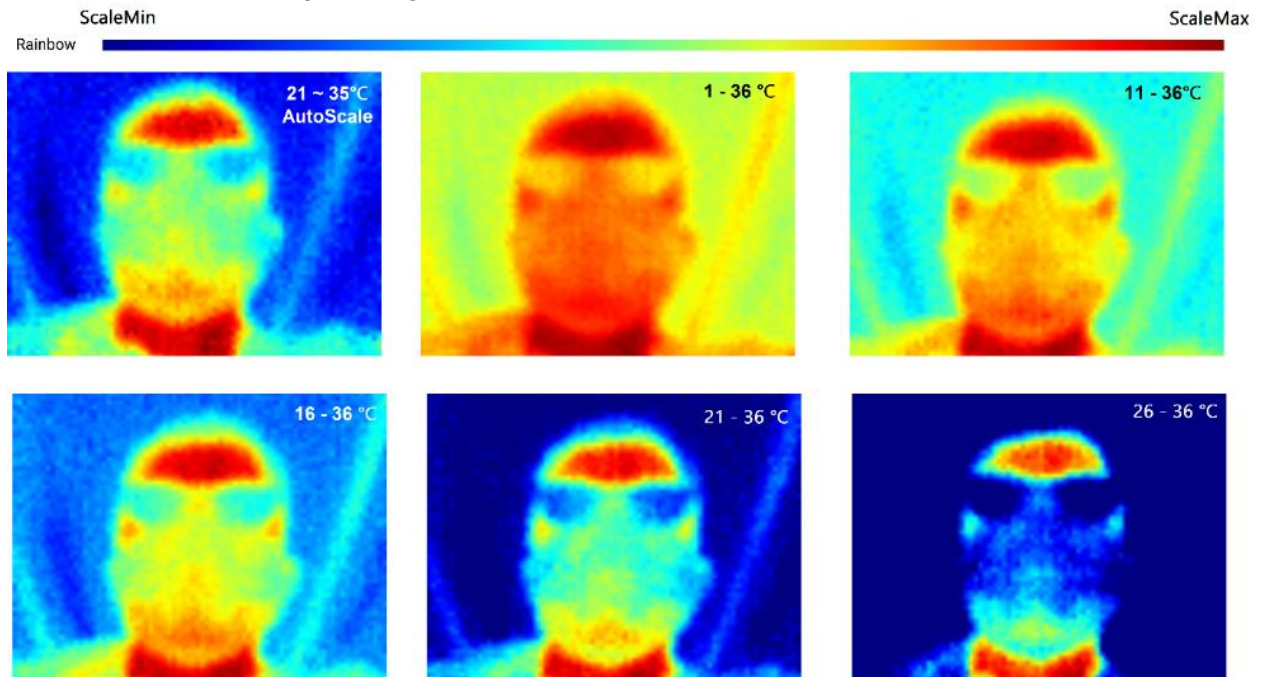
public native void setMinMaxPtrMode(int mode)

Draw min max indicators in thermal image. Set parameter mode 1 - Only maximum indicator is drawn, 2 - Only minimum indicator is drawn, 3 - Both maximum and minimum indicators are drawn, and other - Disabled

public native void setRotation(int rotation)

Rotate a thermal image by 90. Set parameter rotation 1 - Rotate a thermal image by 90 clockwise, 2 - Rotate a thermal image by 90 anti-clockwise, and other - Disabled

The effect of colourmap scaling



SenXorFrameCBT (Core Body Temperature)

The SenXorFrameCBT is trying to develop a human forehead modeling for fever detection in thermal aspect. When it is enabled, the CBT will get the results from Processor and Renderer as the input and try to find contours and calculate its statistic base on mathematical modeling. The statistics of the contours can be get from `getHotSpotStats(roi_idx = 0)` as multiple ROI is available. Default parameter (index 0) refers to full frame ROI, and adding ROI will increase the index in ascendent order.

Constructor

public SenXorFrameCBT()

Create a SenXorFrameCBT object.

Methods

public boolean isDisposed()

Check whether the object is created. Return true if it is not, false otherwise.

public native boolean getIsCBTEnabled();

Check whether CBT is enabled. Return true to if it is enabled, false otherwise.

public native boolean getIsDrawingHotSpots();

Check whether CBT is drawing the hot spots. Return true to if it is, false otherwise.

public native boolean getIsROIExist(int idx);

Check whether the specific index idx of ROI is existed.

public native int addRectROI(int x0, int y0, int width, int height)

Added a rectangle ROI with the top left coordinate (x0 , y0) and the size of of the rectangle (width , height).

public native void setROI(int idx, int x0, int y0, int width, int height)

Set a rectangle ROI with index idx. This required the top left coordinate (x0 , y0) and the size of of the rectangle (width , height) to be specified.

public native void clearROI()

Clear ROI.

public native void clearROI(int roi_idx)

Clear ROI by index idx.

public native void setContouringBlurSize(int _blurSize)

Set the strength (_blurSize) of blurring apply to contour.

public native void setContourThreshold(int _threshold)

Set the threshold (_threshold) of background seperation.

public native void setIsCBTEabled(boolean _newIsCBTEabled)

Enable or disable CBT. Set _newIsCBTEabled to true to enable, false otherwise.

public native void setIsDrawingHotSpots(boolean _newIsDrawingHotSpots)

Enable or disable hot spots drawing. Set _newIsDrawingHotSpots to true to enable, false otherwise.

The data structure of Statistics

centroid_x: The x-coordinate of the contour centroid.

centroid_y: The y-coordinate of the contour centroid.

area: The number of pixels of the contour.

mean: The mean temperature value of the contour area

sdev: The standard deviation of the temperature value inside the contour area

min: The minimum temperature value

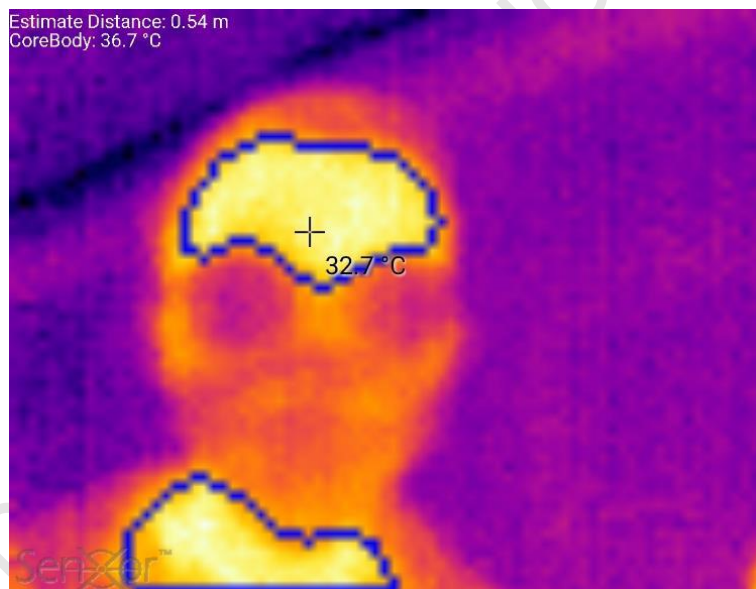
max: The maximum temperature value

spread: The difference between min and max

estimate_distance: The estimated distance between sensor and target

dist_corr_factor: The distance corrected factor for estimate_distance (TBD)

t_core_body: The corrected contour temperature applying human modeling



Appendix

Mapping Temperature to a Colourmap

The output temperature of the Bobcat needs to be mapped to a colourmap before it can be display on a screen.

A typical colourmap can be represented in the form of a lookup table that contains an RGB triplet, with each colour channel having the size of 256 and the entries within each channel bounded between 0 and 255. Therefore, the output temperature needs to be transformed into the corresponding index of the colourmap entry, ranging between 0-255 for zero-indexed and 1-256 for one-indexed.

The temperature to colourmap index transformation can be accomplished via the normalisation and casting of the temperature output through min-max feature scaling, which can restrict the range of the initial dataset between any arbitrary points. Below is the general form of the min-max feature scaling.

$$X' = a + \frac{(X - X_{min})(b - a)}{X_{max} - X_{min}} \quad (1)$$

Where a is the lower bound of the range, b is the upper bound of the range and X is the dataset to be normalised. With a and b equal to 0 and 255 respectively for the case of zero-indexed.

Other than the standard linear conversion, one can also change the dynamic range to compress or stretch the image colour for different temperature range by segmenting the temperature and choosing a and b appropriately.

















Pseudo code in C

```
/* Colormap */
const unsigned char red[256]={255,253,...,255};
const unsigned char green[256]={255,253,...,255};
const unsigned char blue[256]={255,253,...,24};

/* A function to colour a pixel on the display */
void draw(int row, int col, unsigned char r, unsigned char g, unsigned char b);
uint16_t gBuffer[80][62];

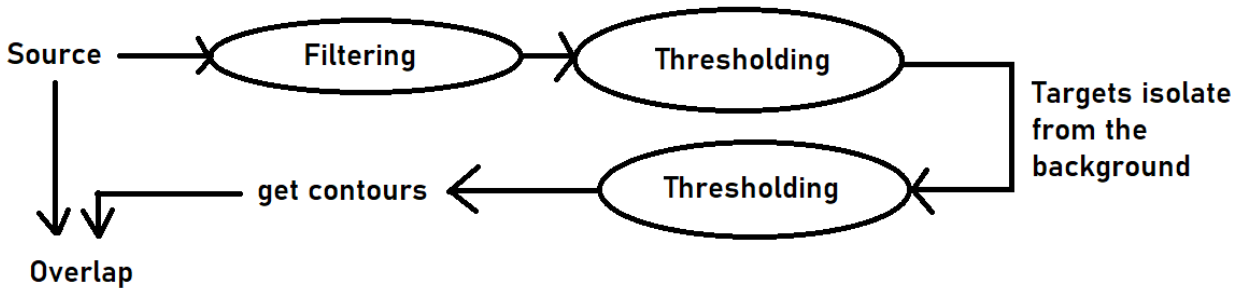
/* min-max feature scaling, gImageMax and gImageMin are the max and min value of gBuffer */
int imageRange = gImageMax - gImageMin;
for (int r = 0; r < IMAGE_ROWS; r++) {
    for (int c = 0; c < IMAGE_COLS; c++) {
        /* Equation 1 with a=0 and b=255 */
        value = (int)(gBuffer[r][c] - gImageMin) * 255 / imageRange;
        draw(r, c, red[value], green[value], blue[value]);
    }
}
```

List of Color Palettes

Heated Iron	
Black Body	
Cool Warm	
Gray Scale	
Iron Glow	
Rainbow	
Autumn	
Bone	
Ocean	
Cool	
Hot	
Magma	
Inferno	
Plasma	
Viridis	
Cividis	

Contouring in Thermal Image

Due to the characteristic of thermal image, contouring becomes one of the most common method to be applied for detection. Here is a simple implementation:



Filtering: Median blurring

Thresholding: Extracting area of interested and masking

Output format

The enum OutputFormat is used to list different stage of the process pipeline. `setOutputFormat()` will change the output on particular stage, default value is RESULT.

```
Enum { SRC = 0, BLUR, THRESHOLD, MASK, RESULT = 4 }
```



Dual camera overlapping

Due to the low resolution of the thermal image, sometimes overlap CMOS image helps to get more details. The concern of overlapping is that the sensor and camera are 2 separate optical system, the image is formed based on its lens, FOV, resolution, view angle to the object, orientation etc.

As to overlap, usually we are going to place one on top of another one directly with 3-D translation, rotation and scaling. However, it works badly if both cameras are not place very close to each other and difficult for the user to adjust, just imagine 2 people looking at the same object at different position. Therefore, we might need another approach called Homography.

Holography is a transformation that maps the points in one image to another. It aligns both images for better overlapping. This methodology only works well for stable object with accurate reference points mapping. If object moves, then transformation has to be recalculated.

Below shows the example of overlapping:

